Ex.No:1.1

LEX Tool

Aim:

Write a lex program whose output is same as input.

Description:

The FLEX tool is a modern version of LEX tool. It takes the input as set of input specifications and gives the output as a complete c program. One of the other tool which is developed in java is JLEX.

Procedure:

1. Create a file with LEX specifications.
2. Save the above file with **.LEX or .l** extension.
3. Compile the program on LEX compiler using command. (lex <filename>).
4. Compile the C program which is generated by LEX tool.

Program:

```
%%
. ECHO;
%%
int yywrap(void) {
return 1;
}
int main(void) {
yylex();
return 0;
}
```

Output:

```
Abcdef
Abcdef
```

Ex.No:1.2

Removing White spaces using LEX Tool

Aim:

Write a lex program which removes white spaces from its input file

Description:

The FLEX tool is a modern version of LEX tool. It takes the input as set of input specifications and gives the output as a complete c program. One of the other tool which is developed in java is JLEX.

Procedure:

1. Create a file with LEX specifications.
2. Save the above file with **.LEX or .l** extension.
3. Compile the program on LEX compiler using command. (lex <filename>).
4. Compile the C program which is generated by LEX tool.

Program:

```
%{
#include<stdio.h>
%}
%%
[' '] {};
%%
main()
{
yylex();
return 0;
}
int yywrap()
{
return 1;
}
```

Output:

```
This is Test File
ThisisTestFile
```

Ex.No:2.1

Identify the patterns

Aim:

Write a lex program to identify the patterns in the input file

Description:

The FLEX tool is a modern version of LEX tool. It takes the input as set of input specifications and gives the output as a complete c program. One of the other tool which is developed in java is JLEX.

Procedure:

1. Create a file with LEX specifications.
2. Save the above file with **.LEX or .l** extension.
3. Compile the program on LEX compiler using command. (lex <filename>).
4. Compile the C program which is generated by LEX tool.

Program:

```
%{
#include<stdio.h>
%}
%%
["int""char""for""if""while""then""return""do"] {printf("keyword : %s\n");}
[*%+\-] {printf("Operator : %s ", yytext);}
[(){};] {printf("Special Character: %s\n", yytext);}
[0-9]+ {printf("Constant : %s\n", yytext);}
[a-zA-Z_][a-zA-Z0-9_]*  {printf("Valid Identifier is : %s\n", yytext);}
^[^a-zA-Z_] {printf("Invalid Indentifier \n");}
%%
```

Output:

```
Valid Identifier is :int
Valid Identifier is : main
Special Character: (
Special Character: )
Special Character: {
Special Character: }
```

Viva Questions:

1. What is the input for LEX tool?

Answer:

Lex specifications (Regular Expressions)

2. Define Lexical Analysis?

Answer:

The lexical analyzer reads the source program one character at a time, carving the source program into a sequence of atomic units called tokens. Identifiers, keywords, constants, operators and punctuation symbols are typical tokens.

3. Write notes on syntax analysis?

Answer:

Syntax analysis is also called parsing. It involves grouping the tokens of the source program into grammatical phrases that are used by the compiler to synthesize output.

4. What is meant by semantic analysis?

Answer:

The semantic analysis phase checks the source program for semantic errors and gathers type information for the subsequent code generation phase. It uses the hierarchical structure determined by the syntax-analysis phase to identify the operators and operand of expressions and statements.

Ex.NO:2.2

Lexical Analyzer

Aim:
To implement a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines.

Description:
Lexical analyzer reads characters from the input, groups them into lexemes and passes the tokens formed by the lexemes. Lexical analysis is the processing of an input sequence of characters to produce, as output, a sequence of symbols called lexical tokens or just "tokens". For example, lexers for many programming languages convert the character sequence `123 abc` into two tokens: `123` and `abc` (white space is not a token in most languages). The purpose of producing these tokens is usually to forward them.

Program:

```
 #include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||
strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||
strcmp("double",str)==0||strcmp("static",str)==0||strcmp("switch",str)==0||
strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str);
}
main()
{
FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
printf("\nEnter the c program");/*gets(st1);*/
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","w");
```

```
while((c=getc(f1))!=EOF)
{
if(isdigit(c))
{
tokenvalue=c-'0';
c=getc(f1);
while(isdigit(c))
{
tokenvalue*=10+c-'0';
c=getc(f1);
}
num[i++]=tokenvalue;
ungetc(c,f1);
}
else if(isalpha(c))
{
putc(c,f2);
c=getc(f1);
while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
{
putc(c,f2);
c=getc(f1);
}
putc(' ',f2);
ungetc(c,f1);
}
else if(c==' '||c=='\t')
printf(" ");
else if(c=='\n')
lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe no's in the program are");
for(j=0;j<i;j++)
printf("%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiers are:");
while((c=getc(f2))!=EOF)
{
if(c!=' ')
str[k++]=c;
else
{
str[k]='\0';
keyword(str);
k=0;
}
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\nSpecial characters are");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d",lineno);
}
```

Output :

```
Enter the C program
a+b*c
Ctrl-D
The no's in the program are:
The keywords and identifiers are:
a is an identifier and terminal
b is an identifier and terminal
c is an identifier and terminal
Special characters are:
+ *
Total no. of lines are: 1*/
```

Viva Questions:

1. What is the function of lexical analyzer?

A) Convert the source file characters into token stream such as keywords, identifiers, constants

2. Which phase of compiler design include lexical analyzer?

A) Analysis phase

3. What is the output of lexical analyzer?

A) Stream of tokens

4. Define pattern?

A) Patterns are the rules for recognizing tokens

Ex.NO:3.1

First and Follow

Aim:
To implement First and Follow of a Grammar

Description:

Rules for calculating FIRST:

1. If X is a terminal symbol then FIRST(X)={X}
2. If $X \to \varepsilon$ is a production then FIRST(X)={ε}

3. If X is a nonterminal and X->Y1Y2…Yk is a production for some k>=1, then place a in First(X) if for some i a is in First(Yi) and ε is in all of First(Y1),…,First(Yi-1) that is Y1…Yi-1 => ε. if ε is in First(Yj) for j=1,…,k then add ε to First(X).

Rules for calculating FOLLOW:

1. If S is the start symbol then add $ to FOLLOW(S)
2. If $A \to \alpha B\beta$ is a production then FOLLOW(B) =FIRST(β)

   if FIRST(β) contains ε then add FOLLOW(A) to FOLLOW(B)

3. If $A \to \alpha B$ is a production FOLLOW(B) = FOLLOW(A).

We apply these rules until nothing more can be added to any follow set.

Program:

PROGRAM FOR COMPUTATION OF FIRST

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void result(char[],char);
int nop;
char prod[10][10];
void main()
{
int i;
char choice,c;
char res1[20];
printf("How many number of productions ? :");
scanf(" %d",&nop);
printf("enter the production string like E=E+T\n");
for(i=0;i<nop;i++)
{
printf("Enter productions Number %d : ",i+1);
scanf(" %s",prod[i]);
do
```

```
{
printf("\n Find the FIRST of :");
scanf(" %c",&c);
FIRST(res1,c);
printf("\n FIRST(%c)= { ",c);
for(i=0;res1[i]!='\0';i++)
printf(" %c ",res1[i]);
printf("}\n");
printf("press 'y' to continue : ");
scanf(" %c",&choice);
}
while(choice=='y'||choice =='Y');
}
void FIRST(char res[],char c)
{
int i,j,k;
char subres[5];
int eps;
subres[0]='\0';
res[0]='\0';
if(!(isupper(c)))
{
result(res,c);
return ;
}
for(i=0;i<nop;i++)
{
if(prod[i][0]==c)
{
if(prod[i][2]=='$')
result(res,'$');
else
{
j=2;
while(prod[i][j]!='\0')
{
eps=0;
FIRST(subres,prod[i][j]);
for(k=0;subres[k]!='\0';k++)
result(res,subres[k]);
for(k=0;subres[k]!='\0';k++)
if(subres[k]=='$')
{
eps=1;
break;
}
if(!eps)
break;
j++;
}
}
}
}
return ;
}
void result(char res[],char val)
{
int k;
for(k=0 ;res[k]!='\0';k++)
if(res[k]==val)
return;
res[k]=val;
res[k+1]='\0';
}
```

Output:

```
How many number of productions ? :8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=$
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i
Find the FIRST of :X
FIRST(X)= { + $ }
press 'y' to continue : Y
Find the FIRST of :F
FIRST(F)= { ( i }
press 'y' to continue : Y
Find the FIRST of :Y
FIRST(Y)= { * $ }
press 'y' to continue : Y
Find the FIRST of :E
FIRST(E)= { ( i }
press 'y' to continue : Y
Find the FIRST of :T
FIRST(T)= { ( i }
press 'y' to continue : N
```

Program to find FOLLOW of a given grammar

```
#include<stdio.h>
#include<string.h>
int nop,m=0,p,i=0,j=0;
char prod[10][10],res[10];
void FOLLOW(char c);
void first(char c);
void result(char);
void main()
{
```

```
int i;
int choice;
char c,ch;
printf("Enter the no.of productions: ");
scanf("%d", &nop);
printf("enter the production string like E=E+T\n");
for(i=0;i<nop;i++)
{
printf("Enter productions Number %d : ",i+1);
scanf(" %s",prod[i]);
}
do
{
m=0;
printf("Find FOLLOW of -->");
scanf(" %c",&c);
FOLLOW(c);
printf("FOLLOW(%c) = { ",c);
for(i=0;i<m;i++)
printf("%c ",res[i]);
printf(" }\n");
printf("Do you want to continue(Press 1 to continue....)?");
scanf("%d%c",&choice,&ch);
}
while(choice==1);
}
void FOLLOW(char c)
{
if(prod[0][0]==c)
result('$');
for(i=0;i<nop;i++)
{
for(j=2;j<strlen(prod[i]);j++)
{
if(prod[i][j]==c)
{
if(prod[i][j+1]!='\0')
first(prod[i][j+1]);
if(prod[i][j+1]=='\0'&&c!=prod[i][0])
FOLLOW(prod[i][0]);
}
}
}
}
void first(char c)
{
int k;
if(!(isupper(c)))
result(c);
for(k=0;k<nop;k++)
{
```

```
if(prod[k][0]==c)
{
if(prod[k][2]=='$')
FOLLOW(prod[i][0]);
else if(islower(prod[k][2]))
result(prod[k][2]);
else
first(prod[k][2]);
}
}
}
void result(char c)
{
int i;
for( i=0;i<=m;i++)
if(res[i]==c)
return;
res[m++]=c;

}
```

Output:

```
Enter the no.of productions: 8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=$
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i
Find FOLLOW of -->X
FOLLOW(X) = { $ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->E
FOLLOW(E) = {$ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->Y
FOLLOW(Y) = { + $ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->T
FOLLOW(T) = { +$ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->F
FOLLOW(F) = { * + $ ) }
Do you want to continue(Press 1 to continue....)?2
```

Viva Questions:

1. Write the algorithm for **FIRST**?
A) To compute FIRST(X) for all grammar symbols X, apply the following rules until no more terminals or e can be added to any FIRST set.
If X is terminal, then FIRST(X) is {X}.
If X->e is a production, then add e to FIRST(X).
If X is nonterminal and X->Y1Y2...Yk is a production, then place a in FIRST(X) if for some i, a is in FIRST(Yi) and e is in all of FIRST(Y1),...,FIRST(Yi-1) that is,
Y1.......Yi- $_1$=*>e. If e is in FIRST(Yj) for all j=1,2,...,k, then add e to FIRST(X).

2. Write the algorithm for **FOLLOW**?
A)Place $ in FOLLOW(S), where S is the start symbol and $ in the input right endmarker.
If there is a production A=>aBs where FIRST(s) except e is placed in FOLLOW(B).
If there is aproduction A->aB or a production A->aBs where FIRST(s) contains e, then everything in FOLLOW(A) is in FOLLOW(B).

.

Ex.No:3.2

LEX Tool

Aim:

To implement the lexical analyzer using JLex, flex or lex or other lexical analyzer generating tools.

Description:

The FLEX tool is a modern version of LEX tool. It takes the input as set of input specifications and gives the output as a complete c program. One of the other tool which is developed in java is JLEX.

Procedure:

a. Create a file with LEX specifications.
b. Save the above file with **.LEX or .l** extension.
c. Compile the program on LEX compiler using command. (lex <filename>).
d. Compile the C program which is generated by LEX tool.

Program:

```
/* program name is lexp.l */
%{
/* program to recognize a c program */
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if  |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto      {printf("\n\t%s is a KEYWORD",yytext);}
"/*"      {COMMENT = 1;}
/*        {printf("\n\n\t%s is a COMMENT\n",yytext) ;}*/
"*/"      {COMMENT = 0;}
/*        printf("\n\n\t%s is a COMMENT\n",yytext);}*/
          {identifier}\(   {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{        {if(!COMMENT) printf("\n BLOCK BEGINS");}
\}        {if(!COMMENT) printf("\n BLOCK ENDS");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\"    {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}
[0-9]+  {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)?   {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}
\(        ECHO;
=         {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
==|
\>    {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc,char **argv)
{
if  (argc > 1)
{
FILE *file;
file = fopen(argv[1],"r");
if(!file)
{
printf("could not open %s \n",argv[1]);
exit(0);
}
yyin = file;
}
yylex();
printf("\n\n");
return 0;
}
int yywrap()
{
return 0;
}
```

Output:

Input:

```
$vi var.c
#include<stdio.h>
main()
{
int a,b;
}
```

Output:

```
$lex lex.l
$cc lex.yy.c
$./a.out var.c
#include<stdio.h> is a PREPROCESSOR DIRECTIVE
FUNCTION
main (
)
```

```
BLOCK BEGINS
int is a KEYWORD
a IDENTIFIER
b IDENTIFIER
BLOCK ENDS
```

Viva Questions:

1. What re the various error-recovery strategies?
A)
a) Panic mode - On discovering this error, the parser discards the input symbols one at a time until one of a designated set of synchronized tokens is found.
b) Phrase level – On discovering an error, a parser perform local correction on the remaining input ; that is , it may replace a prefix or the remaining input by some string that allows the parser to continue.

2.Define ambiguity?
A) A grammar that produces more than one parse tree for some sentence is said to be ambiguous. An ambiguous grammar is one that produces more than one leftmost or more than one right most derivation for some sentence.

3.What is meant by left recursion?
A) A grammar is left recursive if it has a nonterminal A such that there is a derivation A ==> A α for some string α . Top down parsing methods canno handle left-recursion grammars, so a transformation that eliminates left recursion in needed.
Ex:-
E → E +T | T   T → T * F | F   F → (E) | id

Ex.No:4.1

Operator Precedence Parser

Aim:
To implement Operator Precedence parser for a given language.

Description:
Bottom-up parsers for a large class of context-free grammars can be easily developed using operator grammars.Operator grammars have the property that no production rule can have:

- ε at the right side

- two adjacent non-terminals at the right side.

This property enables the implementation of efficient operator-precedence parsers. These parser rely on the following three precedence relations:

Relation Meaning:

a <· b a yields precedence to b
a =· b a has the same precedence as b
a ·> b a takes precedence over b

Program:

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>
 char q[9][9]={
              {'>','>','<','<','<','<','>','<','>'       },
              {'>','>','<','<','<','<','>','<','>'       },
              {'>','>','>','>','<','<','>','<','>'       },
              {'>','>','>','>','<','<','>','<','>'       },
             {'>','>','<','<','<','<','>','<','>'     },
            {'<','<','<','<','<','<','=','<','E'        },
            {'>','>','>','>','>','E','>','E','>'        },



{'>','>','>','>','>','E','>','E','>'},

{'<','<','<','<','<','<','E','<','A' } };
 char s[30],st[30],qs[30]; int top=-1,r=-1,p=0;
 void push(char a)
{
  t

  o

  p

  +

  +

  ;


  s

  t

  [

  t

  o
p
]
=
a
;
}
char pop()
{
char a; a=st[top];
top--;
return a;
}
int find(char a)
{
switch(a)
{
case '+':return 0;
case '-':return 1;
 case '*':return 2;
case'/':return 3;
case '^':return 4;
case '(':return 5;
case ')':return 6;
case 'a':return 7;
case'$':return 8;
default :return -
1;
}
}
void display(char a)
{
```

```
printf("\n Shift %c",a);
}
void display1(char a)
{
if(isalpha(a))
printf("\n Reduce E->%c",a);
else if((a=='+')||(a=='-
')||(a=='*')||(a=='/')||(a=='^'))
printf("\nReduce E->E%cE",a);
else if(a==')')
printf("\n Reduce E->(E)");
}
intrel(char a,char b,char d)
{
if(is
alph
a(a)!
=0)
a='a'
;
if(is
alph
a(b)!
=0)
b='a'
;
if(q[find(a)][find(b)]==d) return 1;
else return 0;
}
void main()
{
char
s[100];
```

```
int i=-1;
clrscr();
printf("\n\t Operator Preceding Parser\n");
printf("\n Enter the Arithmetic Expression End with
$.."); gets(s); push('$');
while(i)
{
if((s[p]=='$')&&(st[top]=='$'))
{
printf("\n\
nAccepted
"); break;
}
else if(rel(st[top],s[p],'<')||rel(st[top],s[p],'='))
{
display(s[p]);
p
u
s
h
(
s
[
p
]
)
;

p
+
+
;
}
else if(rel(st[top],s[p],'>'))
{
do
{
r++;
qs[r]=pop();
display1(qs[r]);
}
while(!rel(st[top],qs[r],'<'));
}
}
}
```

Output:

```
Enter the Arithmetic Expression End with $: a-
(b*c)^d$

Shift a

Reduce E->a

Shift -

Shift (

Shift b

Reduce E->b

Shift *

Shift c

Reduce E->c

Reduce E->E*E

Shift )

Reduce E->(E)

Shift ^

Shift d
```

Reduce E-

>

d

Reduce E->E^E

ReduceE->E-E

Accepted

Viva Questions:

1. What are the two common ways of determining precedence relations should hold between a pair of terminals?

A)

a) Based on associativity and precedence of operators.

b) Construct an unambiguous grammar for the language, a grammar that reflects the correct associativity and precedence in its parse tree.

2. Define operator grammar?

A) A grammar with the property that no production right side is £ or has two adjacent nonterminals are called an operator grammar.

Ex.NO: 4.2

Recursive Descent Parser

Aim:

To implement Recursive Descent Parser for an expression.

Descrption:

A top-down parser that executes a set of recursive procedures to process the input without backtracking is called Recursive Descent Parser. Typically, top-down parsers are implemented as a set of recursive functions that descent through a parse tree for a string. This approach is known as recursive descent parsing.

Program:

Grammar:-

E -> E+T | E-T | T

T -> T*F |

T/F | F F -

> (E) | id

To avoid left recursion we have to write the expression grammer

as follows

E -> TE'

E' -> +TE' | -TE' | $

T -> FT'

T' -> *FT' |

/FT' | $ F -

> (E) | id

```
#include<stdio.h>
char input[20];
int ip=0;
main()
{
clrscr();
printf("Enter the string to be parsed :\t");
scanf("%s",input);
E();
if(input[ip]=='\0')
```

```
printf("Successful Parse");
else
printf("Rejected");
}
E()
{
T();
EPRIME();
}
EPRIME()
{
if(input[ip] == '+' || input[ip] == '-')
{
ip++;
T();
EPRIME();
}
else
return;
}
T()
{
F();
TPRIME();
}
TPRIME()
{
if(input[ip] == '*' || input[ip] == '/')
{
ip++;
F();
TPRIME();
}
else
return;
}
F()
{
if((input[ip] >= 'a' && input[ip] <='z') || (input[ip] >= '0' && input[ip] <= '9'))
ip++;
else
if(input[ip] == '(' )
{
ip++;
E();
if(input[ip] == ')' )
ip++;
else
error();
}
else
error();
}
error()
{
printf("\n The String is rejected ");
exit(0);
}
```

Output:

```
Enter the string to be parsed : 5+3*(a-b)/4
Successful Parse
```

Viva Questions:

1. What is meant by recursive-descent parser?

A) A parser that uses a set of recursive procedures to recognize its input with no backtracking is called a recursive-descent parser. To avoid the necessity of a recursive language, we shall also consider a tabular implementation of recursive descent called predictive parsing.

2. Define top down parsing?

A) It can be viewed as an attempt to find the left most derivation for an input string. Itcan be viewed as attempting to construct a parse tree for the input starting from the root and creating the nodes of the parse tree in preorder.

3. What are the possibilities of non-recursive predictive parsing?

A) If X = a = \$ the parser halts and announces successful completion of parsing.

If X = a = \$ the parser pops X off the stack and advances the input pointer to the next symbol

If X is a nonterminal, the program consults entry M[X,a] of the parsing table M. This entry will be either an X-production of the grammar or an error entry. If M[X,a] holds a production rule $X \rightarrow Y_1Y_2...Y_k$, it pops X from the stack and pushes $Y_k, Y_{k-1}, ..., Y_1$ into the stack.

EX.NO:5.1

LL(1) Parser

Aim:

To implement LL(1) Parser for an expression.

Description:

A grammar whose parsing table has no multiply defined entries is said to be LL(1). It can be shown that the above algorithm can be used to produce for every LL(1) grammar G a parsing tab

le M that parses all and only the sentences of G. LL(1) grammars have several distinctive properties. No ambiguous or left recursive grammar can be LL(1). There remains a question of what should be done in case of multiply defined entries. One easy solution is to eliminate all left recursion and left factoring, hoping to produce a grammar which will produce no multiply defined entries in the parse tables. Unfortunately there are some grammars which will give an LL(1) grammar after any kind of alteration. In general, there are no universal rules to convert multiply defined entries into single valued entries without affecting the language recognized by the parser.

Program:

```
#include<stdio.h>
int stack[20],top=-1;
void push(int item)
{
if(top>=20)
{
printf("STACK OVERFLOW");
exit(1);
}
stack[++top]=item;
}
int pop()
{
int ch;
if(top<=-1)
{
printf("underflow");
exit(1);
}
ch=stack[top--];
return ch;
}
char convert(int item)
```

```
{
char ch;
switch(item)
{
case 0:return('E');
case 1:return('e');
case 2:return('T');
case 3:return('t');
case 4:return('F');
case 5:return('i');
case 6:return('+');
case 7:return('*');
case 8:return('(');
case 9:return(')');
case 10:return('$');
}
}
void main()
{
int m[10][10],i,j,k;
char ips[20];
int ip[10],a,b,t;
m[0][0]=m[0][3]=21;
m[1][1]=621;
m[1][4]=m[1][5]=-2;
m[2][0]=m[2][3]=43;
m[3][1]=m[3][4]=m[3][5]=-2;
m[3][2]=743;
m[4][0]=5;
m[4][3]=809;
printf("\n enter the input string:");
scanf("%s",ips);
for(i=0;ips[i];i++)
{
switch(ips[i])
{
case 'E':k=0;break;
case 'e':k=1;break;
case 'T':k=2;break;
case 't':k=3;break;
case 'F':k=4;break;
case 'i':k=5;break;
case '+':k=6;break;
case '*':k=7;break;
case '(':k=8;break;
case ')':k=9;break;
case '$':k=10;break;
}
ip[i]=k;
}
ip[i]=-1;
push(10);
push(0);
i=0;
printf("\tstack\t input \n");
while(1)
{
printf("\t");
for(j=0;j<=top;j++)
printf("%c",convert(stack[j]));
printf("\t\t");
for(k=i;ip[k]!=-1;k++)
printf("%c",convert(ip[k]));
printf("\n");
if(stack[top]==ip[i])
{
if(ip[i]==10)
{
printf("\t\t SUCCESS");
return;
}
else
{
top--;
i++;
}
}
else if(stack[top]<=4&&stack[top]>=0)
{
a=stack[top];
b=ip[i]-5;
t=m[a][b];
top--;
while(t>0)
{
push(t%10);
t=t/10;
}
}
else
{
printf("ERROR");
return;
}
}
}
```

Output:

```
    enter the string:i+(i*i)$
            stack           input
```

```
$E              i+(i*i)$
$eT             i+(i*i)$
$etF            i+(i*i)$
$eti            i+(i*i)$
$et              +(i*i)$
$e               +(i*i)$
$eT+             +(i*i)$
$eT               (i*i)$
$etF              (i*i)$
$et)E(            (i*i)$
$et)E              i*i)$
$et)eT             i*i)$
$et)etF            i*i)$
$et)eti            i*i)$
$et)et              *i)$
$et)etF*            *i)$
$et)etF              i)$
$et)eti              i)$
$et)et                )$
$et)e                 )$
$et)                  )$
$et                    $
$e                     $
$                      $
     SUCCESS
```

<u>Viva Questions:</u>

1. What are the properties of LL (1) grammar?

A) A grammar G is LL (1) if and only if, whenever A->α / β are two distinct productions of G of the following conditions

a) For no terminal 'a' do both α and β derive strings beginning with α.
b) At most one of α and β can derive the empty string.
c) If β* e then α does not derive any string beginning with a terminal in FOLLOW (A).

2. What is LL (1) grammar?

A) A grammar is LL(1) grammar if and only if its M-table has no entries that are multiply defined.

3. What are the possible error recovery actions in lexical analysis:

A)

a) Deleting an extraneous character

b) Inserting a missing character

c) Replacing an incorrect character by a correct character

d) Transposing two adjacent characters

Ex.No:5.2

LALR Bottom up Parser

Aim:

To implement LALR bottom up parser for the given language.

Description:

LALR parsing is one of the bottom-up parsing technique. In this parsing the parse table is used with input. The parse table doesn't contain multiple entries. So the There is no backtracking in LALR parser. LALR parser is the efficient bottom-up parser.

Program:

```
#include<stdio.h>
```

```
int st[20],top=-1;
char input[20];
int encode(char ch)
{
switch(ch)
{
case 'i':return 0;
case '+':return 1;
case '*':return 2;
case '(':return 3;
case ')':return 4;
case '$':return 5;
case 'E':return 6;
case 'T':return 7;
case 'F':return 8;
}
return -1;
}
char decode(int n)
{
switch(n)
{
case 0:return('i');
case 1:return('+');
case 2:return('*');
case 3:return('(');
case 4:return(')');
case 5:return('$');
case 6:return('E');
case 7:return('T');
case 8:return('F');
}
return 'z';
}
void push(int n)
{
st[++top]=n;
}
int pop()
{
return(st[top--]);
}
void display(int p,char *ptr)
{
int l;
for(l=0;l<=top;l++)
{
if(l%2==1)
printf("%c",decode(st[l]));
else
printf("%d",st[l]);
}
printf("\t");
for(l=p;ptr[l];l++)
printf("%c",ptr[l]);
printf("\n");
}
void main()
{
char t1[20][20],pr[20][20],xy;
int inp[20],t2[20][20],gt[20][20];
int i,k,x,y,tx=0,ty=0,len;
clrscr();
strcpy(pr[1],"E E+T");
strcpy(pr[2],"E T");
strcpy(pr[3],"T T*F");
strcpy(pr[4],"T F");
strcpy(pr[5],"F (E)");
strcpy(pr[6],"F i");
t2[2][1]=t2[2][4]=t2[2][5]=2;
t2[3][1]=t2[3][2]=t2[3][4]=t2[3][5]=4;
t2[5][1]=t2[5][2]=t2[5][4]=t2[5][5]=6;
t2[9][1]=t2[9][4]=t2[9][5]=1;
t2[10][1]=t2[10][2]=t2[10][4]=t2[10][5]=3;
t2[11][2]=t2[11][1]=t2[11][4]=t2[11][5]=5;
t1[2][1]=t1[2][4]=t1[2][5]='r';
t1[3][1]=t1[3][2]=t1[3][4]='r';
t1[3][5]=t1[5][1]=t1[5][2]='r';
t1[5][4]=t1[5][5]=t1[9][1]=t1[9][4]='r';
t1[9][5]=t1[10][1]=t1[10][2]=t1[10][4]=t1[10][5]='r';
t1[11][1]=t1[11][4]=t1[11][2]=t1[11][5]='r';
t1[0][0]=t1[4][0]=t1[6][0]=t1[7][0]=t1[0][3]=t1[4][3]=t1[6][3]='s';
t1[2][2]=t1[9][2]=t1[8][4]=t1[1][1]=t1[8][1]=t1[7][3]='s';
t1[1][5]='a';
t2[0][0]=t2[4][0]=t2[6][0]=t2[7][0]=5;
t2[0][3]=t2[4][3]=t2[6][3]=t2[7][3]=4;
t2[2][2]=t2[9][2]=7;
t2[8][4]=11;
t2[1][1]=t2[8][1]=6;
gt[0][6]=1;
gt[0][7]=gt[4][7]=2;
gt[0][8]=gt[4][8]=gt[6][8]=3;
gt[4][6]=8;gt[6][7]=9;gt[7][8]=10;
printf("enter string:");
scanf("%s",input);
for(k=0;input[k];k++)
{
inp[k]=encode(input[k]);
if(input[k]<0||inp[k]>5)
printf("\n error in input");
}
push(0);
```

```
i=0;
while(1)
{
x=st[top];y=inp[i];
display(i,input);
if(t1[x][y]=='a')
{
printf("string is accepted \n");
exit(0);
}
else if(t1[x][y]=='s')
{
push(inp[i]) ;
push(t2[x][y]);
i++;
}
else if(t1[x][y]=='r')
{
len=strlen(pr[t2[x][y]])-2;
xy=pr[t2[x][y]][0];
ty=encode(xy);
for(k=1;k<=2*len;k++)
pop();
tx=st[top];
push(ty);
push(gt[tx][ty]);
}
else
printf("\n error in parsing");
}
}
```

Output:

```
enter string:i*(i+i)$
0               i*(i+i)$
0i5             *(i+i)$
0F3             *(i+i)$
0T2             *(i+i)$
0T2*7           (i+i)$
0T2*7(4         i+i)$
0T2*7(4i5       +i)$
0T2*7(4F3       +i)$
0T2*7(4T2       +i)$
0T2*7(4E8       +i)$
0T2*7(4E8+6     i)$
```

```
0T2*7(4E8+6i5   )$
0T2*7(4E8+6F3   )$
0T2*7(4E8+6T9   )$
0T2*7(4E8       )$
0T2*7(4E8)11    $
0T2*7F10        $
0T2             $
0E1             $
string is accepted
```

Viva Questions:

1. Define LALR grammar?

A) Last parser construction method, the LALR technique. This method is often used in practice because the tables obtained by it are considerably smaller than the canonical LR tables, yet most common syntactic constructs of programming language can be expressed conveniently by an LALR grammar. If there are no parsing action conflicts, then the given grammar is said to be an LALR (1) grammar. The collection of sets of items constructed is called LALR (1) collections.

2. What are the problems in top down parsing?

A)

a) Left recursion.

b) Backtracking.

c) The order in which alternates are tried can affect the language accepted.

3. Define recursive-descent parser?

A) A parser that uses a set of recursive procedures to recognize its input with nonbacktracking is called a recursive-descent parser. The recursive procedures can be quite easy to write.

Ex.NO:6.1

Loop Unrolling

Aim:
Write a Program to perform Loop Unrolling.

Description:

Loop unrolling, also known as loop unwinding, is a loop transformation technique that attempts to optimize a program's execution speed at the expense of its binary size, which is an approach known as the space-time tradeoff. The transformation can be undertaken manually by the programmer or by an optimizing compiler.

Program:

```
#include<stdio.h>
#define TOGETHER (8)
int main(void)
{
int i = 0;
int entries = 50;                          /* total number to process     */
int repeat;                               /* number of times for while.. */
int left = 0;                             /* remainder (process later)   */
/* If the number of elements is not be divisible by BLOCKSIZE,  */
/* get repeat times required to do most processing in the while loop   */
repeat = (entries / TOGETHER);                  /* number of times to repeat   */
left   = (entries % TOGETHER);                  /* calculate remainder  */
/* Unroll the loop in 'bunches' of 8   */
while (repeat--)
{
```

```
printf("process(%d)\n", i    );
printf("process(%d)\n", i + 1);
printf("process(%d)\n", i + 2);
printf("process(%d)\n", i + 3);
printf("process(%d)\n", i + 4);
printf("process(%d)\n", i + 5);
printf("process(%d)\n", i + 6);
printf("process(%d)\n", i + 7);
/* update the index by amount processed in one go    */
i += TOGETHER;
}
/* Use a switch statement to process remaining by jumping to the case label   */
/* at the label that will then drop through to complete the set*/
switch (left)
{
case 7 : printf("process(%d)\n", i + 6);     /* process and rely on drop through */
case 6 : printf("process(%d)\n", i + 5);
case 5 : printf("process(%d)\n", i + 4);
case 4 : printf("process(%d)\n", i + 3);
case 3 : printf("process(%d)\n", i + 2);
case 2 : printf("process(%d)\n", i + 1);      /* two left  */
case 1 : printf("process(%d)\n", i);          /* just one left to process      */
case 0 : ;                                    /* none left   */
}
}
```

<u>Output:</u>

```
 Process 1;
 Process 2;
 Process 3;
 Process 4;
 Process 5;
 Process 6;
 Process 7;
 Process 8;
 Process 9;

 Process 7;
 Process 6;
 Process 5;
 Process 4;
 Process 3;
 Process 2;
```

Process 1;

Viva Questions:

1.What is Loop Unrolling?

A) Loop unrolling, also known as loop unwinding, is a loop transformation technique that attempts to optimize a program's execution speed at the expense of its binary size, which is an approach known as the space-time tradeoff.

2. What are the representations of three-address statements?
A) A three address statement is an abstract form of intermediate code. There are three representation are available. They are
a) Quadruples
b) Triples
c) Indirect triples

3. Write notes on control stack?
A) A control stack is to keep track of live procedure activations. The idea is to push the node for activation onto the control stack as the activation begins and to pop the node when the activation ends.

4. Write the scope of a declaration?
A) A portion of the program to which a declaration applies is called the scope of that declaration. An occurrence of a name in a procedure is said to be local to procedure if it is in the scope of a declaration within the procedure; otherwise the occurrence is said to be nonlocal.

Ex.NO:6.2

Loop Unrolling

Aim:

Write a program for constant propagation

Description:

Constant propagation is a compiler optimization technique that aims to replace variables with their constant values if they are known at compile-time. This can help improve the performance of the generated code by eliminating unnecessary variable accesses.

Program:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<conio.h>
void input();
void output();
void change(int p,char *res);
void constant();
struct expr
{
char op[2],op1[5],op2[5],res[5];
int flag;
}arr[10];
int n;
void main()
{
clrscr();
input();
constant();
output();
getch();
}
void input()
{
int i;
printf("\n\nEnter the maximum number of  expressions : ");
scanf("%d",&n);
printf("\nEnter the input : \n");
for(i=0;i<n;i++)
{
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);
arr[i].flag=0;
}
}
void constant()
{
int i;
int op1,op2,res;
char op,res1[5];
for(i=0;i<n;i++)
{
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=")==0) /*if both digits,
store them in variables*/
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];
switch(op)
{
case '+':
res=op1+op2;
break;
case '-':
res=op1-op2;
break;
case '*':
res=op1*op2;
break;
case '/':
res=op1/op2;
break;
case '=':
res=op1;
break;
}
sprintf(res1,"%d",res);
arr[i].flag=1; /*eliminate expr and replace any operand below that uses result of this expr */
change(i,res1);
}
}
}
void output()
{
int i=0;
printf("\nOptimized code is : ");
for(i=0;i<n;i++)
{
if(!arr[i].flag)
{
printf("\n%s %s %s %s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
}
}
void change(int p,char *res)
```

```
{
int i;
for(i=p+1;i<n;i++)
{
if(strcmp(arr[p].res,arr[i].op1)==0)
strcpy(arr[i].op1,res);
else if(strcmp(arr[p].res,arr[i].op2)==0)
strcpy(arr[i].op2,res);
}
}
```

Output:

```
Optimized code is :
+ 3 b t1
+ 3 c t2
+ t1 t2 t3
```

FAMILIARIZATION WITH RATIONAL ROSE OR UMBRELLO:

INTRODUCTION TO UML
UNIFIED MODELING LANGUAGE:

The UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML gives you a standard way to write a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components.

An Overview of UML

- The Unified Modeling Language is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system.
- The UML is appropriate for modeling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems. It is a very expressive language, addressing all the views needed to develop and then deploy such systems.

  The UML is a language for

  - Visualizing
  - Specifying
  - Constructing
  - Documenting

- **Visualizing**The UML is more than just a bunch of graphical symbols. Rather, behind each symbol in the UML notation is a well-defined semantics. In this manner, one developer can write a model in the UML, and another developer, or even another tool, can interpret that model unambiguously

- **Specifying** means building models that are precise, unambiguous, and complete.

- **Constructing** the UML is not a visual programming language, but its models can be directly connected to a variety of programming languages.

- **<u>Documenting</u>** a healthy software organization produces all sorts of artifacts in addition to raw executable code. These artifacts include
    - Requirements
    - Architecture
    - Design
    - Source code
    - Project plans
    - Tests
    - Prototypes
    - Releases

To understand the UML, you need to form a **conceptual model of the language**, and this requires learning three major elements:

1. Things
2. Relationships
3. Diagrams

Things in the UML

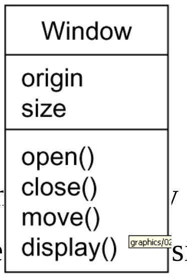
There are four kinds of things in the UML:

Structural things
Behavioral things
Grouping things
Annotational things

Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

1. Classes
2. Interfaces
3. Collaborations
4. Use cases
5. Active classes
6. Components
7. Nodes

Class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations.

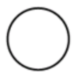


Interface

Interface is a collection of operation / a service of a class or component.

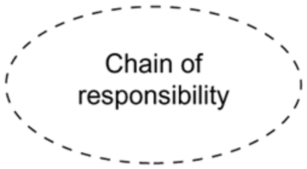
An interface therefore describes the sible behavior of that element.

An interface might represent the complete behavior of a class or component or only a part of that behavior.An interface is rendered as a circle together with its name. An interface rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface



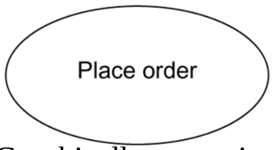
Collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. Therefore, collaborations have structural, as well as behavioral, dimensions. A given class might participate in several collaborations.

Graphically, a collaboration is rendered as an ellipse with dashed lines, usually including only its name

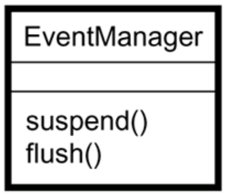


Usecase

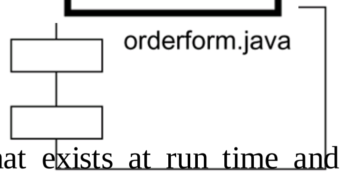
- Use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor
- Use case is used to structure the behavioral things in a model.
- A use case is realized by collaboration. Graphically, a use case is rendered as an ellipse with solid lines, usually including only its name



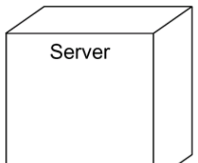
Active class is just like a clas ts represent elements whose behavior is concurrent with other elements. Graphically, an active class is rendered just like a class, but with heavy lines, usually including its name, attributes, and operations



Component is a physical and rep ystem that conforms to and provides the realization of a set of interfaces. Gr ent is rendered as a rectangle with tabs



Node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. Graphically, a node is rendered as a cube, usually including only its name



Behavioral Things are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space. In all, there are two primary kinds of behavioral things

Interaction

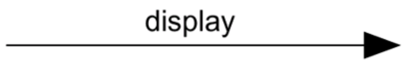
state machine

Interaction

Interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose

An interaction involves a number of other elements, including messages, action sequences and links

Graphically a message is rendered as a directed line, almost always including the name of its operation

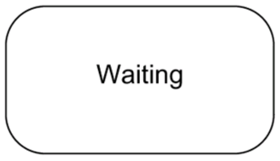


<u>State Machine</u>

State machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events

State machine involves a number of other elements, including states, transitions, events and activities

Graphically, a state is rendered as a rounded rectangle, usually including its name and its substates

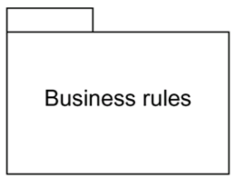


<u>Grouping Things:-</u>

1. are the organizational parts of UML models. These are the boxes into which a model can be decomposed
2. There is one primary kind of grouping thing, namely, packages.

<u>Package:-</u>

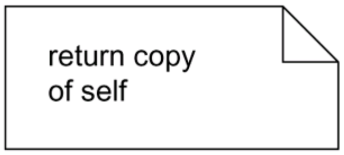
- A package is a general-purpose mechanism for organizing elements into groups. Structural things, behavioral things, and even other grouping things may be placed in a package
- Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents



Annotational things are the explanatory parts of UML models. These are the comments you may apply to describe about any element in a model.

A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.

Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment



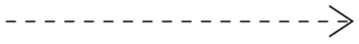
Relationships in the UML: There are four kinds of relationships in the UML:

1. Dependency
2. Association
3. Generalization
4. Realization

Dependency:-

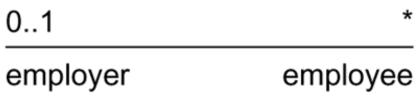
Dependency is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing

Graphically a dependency is rendered as a dashed line, possibly directed, and occasionally including a label

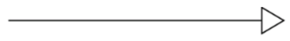


Association is a structural relationship that describes a set of links, a link being a connection among objects.

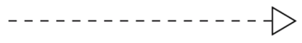
Graphically an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names



<u>Aggregation</u> is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent



<u>Realization</u> is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. Graphically a realization relationship is rendered as a cross between a generalization and a dependency relationship



Diagrams in the UML

Diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships).

In theory, a diagram may contain any combination of things and relationships.

For this reason, the UML includes nine such diagrams:

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram
- Component diagram
- Deployment diagram

<u>Class diagram</u>

A class diagram shows a set of classes, interfaces, and collaborations and their relationships.

Class diagrams that include active classes address the static process view of a system.

<u>Object diagram</u>

- Object diagrams represent static snapshots of instances of the things found in class diagrams
- These diagrams address the static design view or static process view of a system
- An object diagram shows a set of objects and their relationships

Use case diagram

- A use case diagram shows a set of use cases and actors and their relationships
- Use case diagrams address the static use case view of a system.
- These diagrams are especially important in organizing and modeling the behaviors of a system.

Interaction Diagrams

Both sequence diagrams and collaboration diagrams are kinds of interaction diagrams

Interaction diagrams address the dynamic view of a system

Asequence diagram is an interaction diagram that emphasizes the time-ordering of messages

A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages

Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other

Statechart diagram

- A statechart diagram shows a state machine, consisting of states, transitions, events, and activities
- Statechart diagrams address the dynamic view of a system
- They are especially important in modeling the behavior of an interface, class, or collaboration and emphasize the event-ordered behavior of an object

Activity diagram

An activity diagram is a special kind of a statechart diagram that shows the flow from activity to activity within a system

Activity diagrams address the dynamic view of a system

They are especially important in modeling the function of a system and emphasize the flow of control among objects

Component diagram

- A component diagram shows the organizations and dependencies among a set of components.
- Component diagrams address the static implementation view of a system
- They are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations

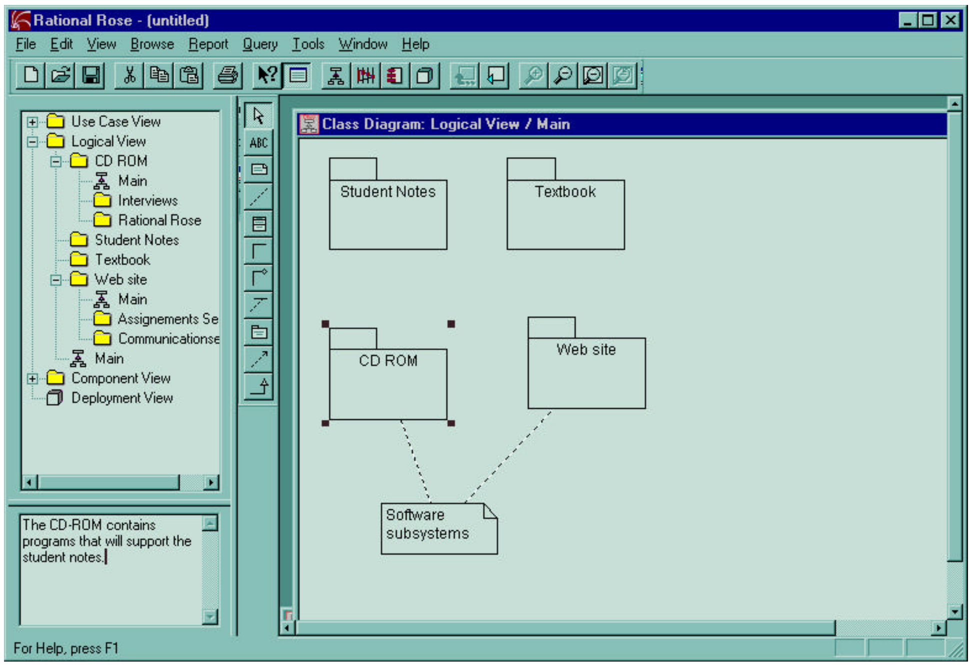
Deployment diagram

- A deployment diagram shows the configuration of run-time processing nodes and the components that live on them

Deployment diagrams address the static deployment view of an architecture

An Introduction to Rational Rose

In this course you will be learning the Unified Modeling Language (UML), which is a special notation forsystems analysis and design. Creating UML diagrams requires a diagramming tool. The tool you will be using for this purpose is a student version of Rational Rose. Rational Rose is a sophisticated CASE tool with a number of automated features, including code generation and reverse engineering. The version you will use in this course is not the most recent version and there are limitations on the size of the diagrams you can save.



The CD-ROM contains programs that will support the student notes.

For Help, press F1

1. Browser Window

This presents a hierarchical view of the analysis and design model, including all the diagrams and all theindividual elements that make up a diagram.

2. Drawing Tools

This tool presents a set of icons that indicate the different elements that can be added to a diagram. The elements that can be used will change, depending on the type of diagram being created. Different diagram types have different sets of icons. If you were creating a different diagram type, you would see a different set of icons. The above example is a class diagram in logical view.

3. Diagram Window

This is where the diagram is actually created. You will see that the diagram shown in the drawing window on Figure 1 represents a high-level model of this course. Course content can be seen as a system composed of four interacting subsystems, two of which involve software. We have used the Package element to represent the subsystems, and the Note element to indicate which packages contain software.

4. Documentation Window

It is strongly recommended that each element added to a diagram have documentation to accompany it. To add documentation, right click on the element, select specification, and fill in the documentation field. The documentation will then be shown in the documentation window each time the mouse is clicked on the element.Documentation can also be added directly to the documentation window.

Views in UML/Rational Rose

There are four views for a model created in Rational Rose, each representing the system from a different point ofview.

The Use Case View

The use case view contains the diagrams used in analysis (use case, sequence, and collaboration), and all the elements that comprise these diagrams (e.g., actors). More recent versions of Rational Rose also allow for additional documentation in the form of word-processed documents and/or URLs to Web-based materials. The purpose of the use case view is to envisage what the system must do, without dealing with the specifics of how it will be implemented.

Logical View

The logical view contains the diagrams used in object design (class diagrams and state transition diagrams). It offers a detailed view of how the system envisaged in the use case view will be implemented. The basic element in this view is the *class*, which includes an outline of its attributes and operations. This directly corresponds to a class created in your chosen implementation language. From the logical view, skeletal code can be generated for implementation into a computer language. More recent versions of Rational Rose not only can generate skeletal code for Visual C++, Visual Java, or Visual BASIC, but also reverse engineer programs created in these languages into Rational Rose models. This allows existing components to be included in documented models, if there is access to the source code. In addition, changes that need to be made during implementation can be reflected in the documentation of the design model.

Component View

The component view is a step up from the logical view and contains diagrams used in system design (component diagrams). This includes information about the code libraries, executable programs, runtime libraries, and other software components that comprise the completed systems. Components can be pre-existing; for example, a Windows program in Visual C++ will utilize Microsoft Foundation Class to provide the framework for the Windows interface. Components that do not exist and need to be created by the developers will have to be designed in the logical view.

Deployment View

The deployment view illustrates how the completed system will be physically deployed. This view is necessary for complex applications in which a system will have different components located on different machines. For example, interface components may be located on a user machine while other components may be located on a network server.

LIBRARY MANAGEMENT SYSTEM

Problem Definition:

This system can manage all the happenings of the Library. Book transactions including Book Registration, Students Registration, Book Issuing, Current Status of a particular books etc. can be very easily handled by this module. Overall this system can be very helpful and it can make things easier.

Requirements:

- This software will allow members to register or administrator can add users.
- Only registered members will be allowed to lend an item from the system
- User can add items (Books, CD etc) to the system
- System will allow searching for items in the system based on Author name, book name, user name etc

a) Identify and analyze events

Books:

- ➢ Search Books
- ➢ View Books
- ➢ Add books
- ➢ Remove Books

Transaction:

- ➢ Return Book
- ➢ Search Book
- ➢ Search Student
- ➢ Borrow Book

Register:

- ➢ Register-Student
- ➢ Register-Staff

Report:

- ➢ Borrower List
- ➢ List of Available Books

Administrator:

- ➢ Add Books
- ➢ Manage User Accounts
- ➢ Update Books
- ➢ Remove Book

View/Edit:

- View Student
- View Staff
- Edit Student
- Edit Staff
- View Books
- Edit Books

Reader services:

- References
- generals

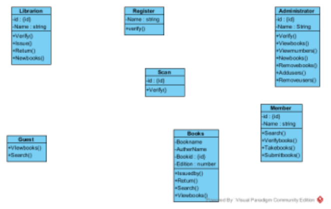
b) Identify Use cases

- Scan id
- Register
- View Books
- View Members
- Search Books
- Issue Books
- Return Books
- Add/Remove Books
- Add/Remove Members
- Check issued book is Issued Book / Reference Book
- Check the Member Registered Member or not

c) Develop event table

	Student	Librarian	Admin	Data base
Register	+	+	+	
View Books	*	*	*	*
Search Books	*	*	*	*
Add Books		*		*
Issue Books	*	*		

d) Identify & analyze domain classes



e) Develop CRUD matrix to represent relationships between use cases and problem domain classes

Data Entity	CRUD	Resulting Use case
Student	Create Read/Report Update Delete	Add new Student Find new Student Generate Student Reserve list Update Student Information Delete Passed out student
Book	Create Read/Report Update Delete	Add new Book View Books Search Books Update Book list Delete Books list
Librarian	Create Read/report Update Delete	Add new Books View Books Search Books Update Student/Book list Delete Student/Book list

Aim: To draw use case diagram for library application.

Software used: Rational Rose

Procedure: Draw and drop the actors and usecase from browser window into the diagram window. Associate the usecases and actors.

Description: An usecase diagram is a description of set of sequence of actions including variants that the system performs which yields an observable result of value to an actor.

The usecase diagram of library system contains two actions majorly. The librarian and the member.

The member may be either a student or staff. The member interacts with a set of usecases where he can enter library, register details, search, barrow,reserve items and also can pay the fine

The librarian checks the details, issues books, placing or cancelling orders and also he collects the fine.

Result: We have successfully worked with the unified library application. Use case diagram for library application in use case

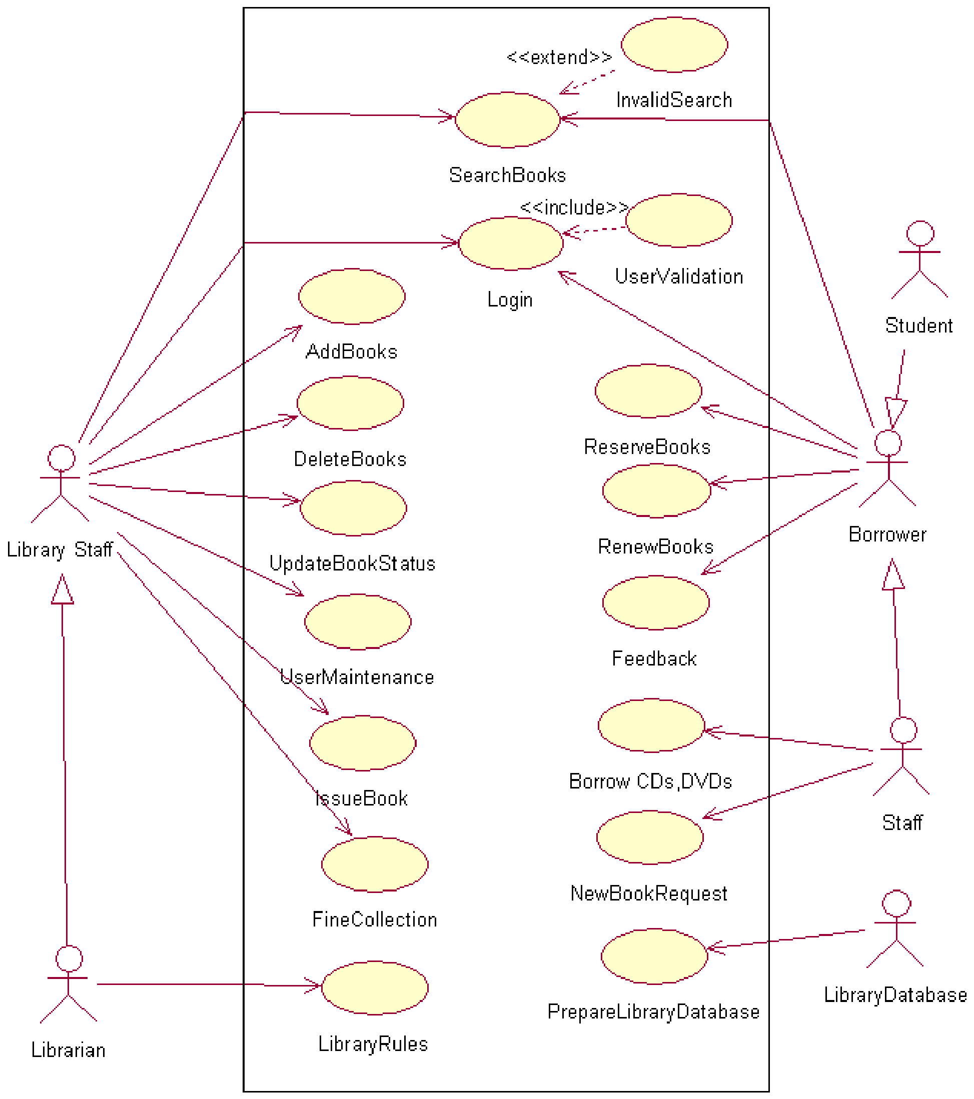


Fig: Use case diagram

Actors

1. LibraryStaff
2. Librarian
3. Borrower
4. Staff
5. Student
6. LibraryDatabase

Use Cases

1. Searchbooks
2. Login
3. AddBooks
4. DeleteBooks
5. UpdateBookStatus
6. UserMaintenance
7. IssueBook
8. FineCollection
9. LibraryRules
10. ReserveBooks
11. RenewBooks
12. FeedBack
13. BorrowItems
14. NewBookRequest
15. PrepareLibraryDatabase

Search Use Case

A search use case is started when the user, either library staff or borrower initiates the operation.

Issue Use Case

A issue operation is invoked by librarian. After validating user and searching is successful the book is issued with due date.

Return Use Case

A return operation is initiated by user. The librarian has to updateBookStatus and FineCollection if necessary

Renew Use Case

A renew operation is also initiated by user. The librarian has to update Book Status

Aim : The activity diagram for library application.

Software used: Rational Rose

Procedure: Create a new package from the database and name it as a activity diagram from the toolbar, select the required tools and make corresponding connections.

Description:

- Select the object that have high level responsibilities.
- These objects may be real or abstract. In either case, create a swim lane for each important object.
- Identify the precondition of initial state and post conditions of final state.
- Beginning at initial state, specify the activities and actions and render them as activity states or action states.
- For complicated actions, or for a set of actions that appear multiple times, collapse these states and provide separate activity diagram.
- Render the transitions that connect these activities and action states.
- Start with sequential flows; consider branching, fork and joining.
- Adorn with notes tagged values and so on.

Result: We have successfully worked with the activity diagram for unified library application.

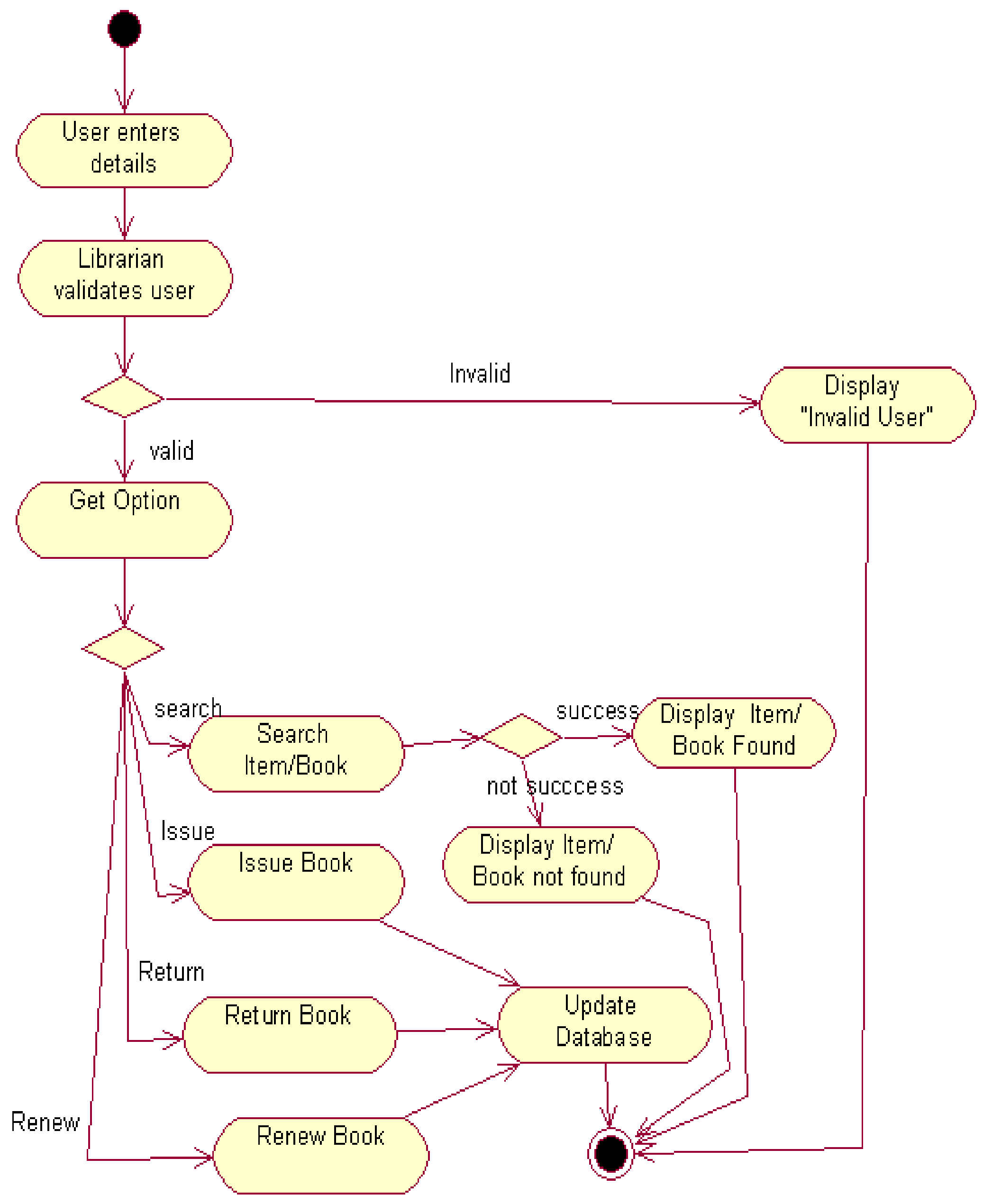


Fig: Activity diagram

Aim: Construct Use case diagram using relation ships

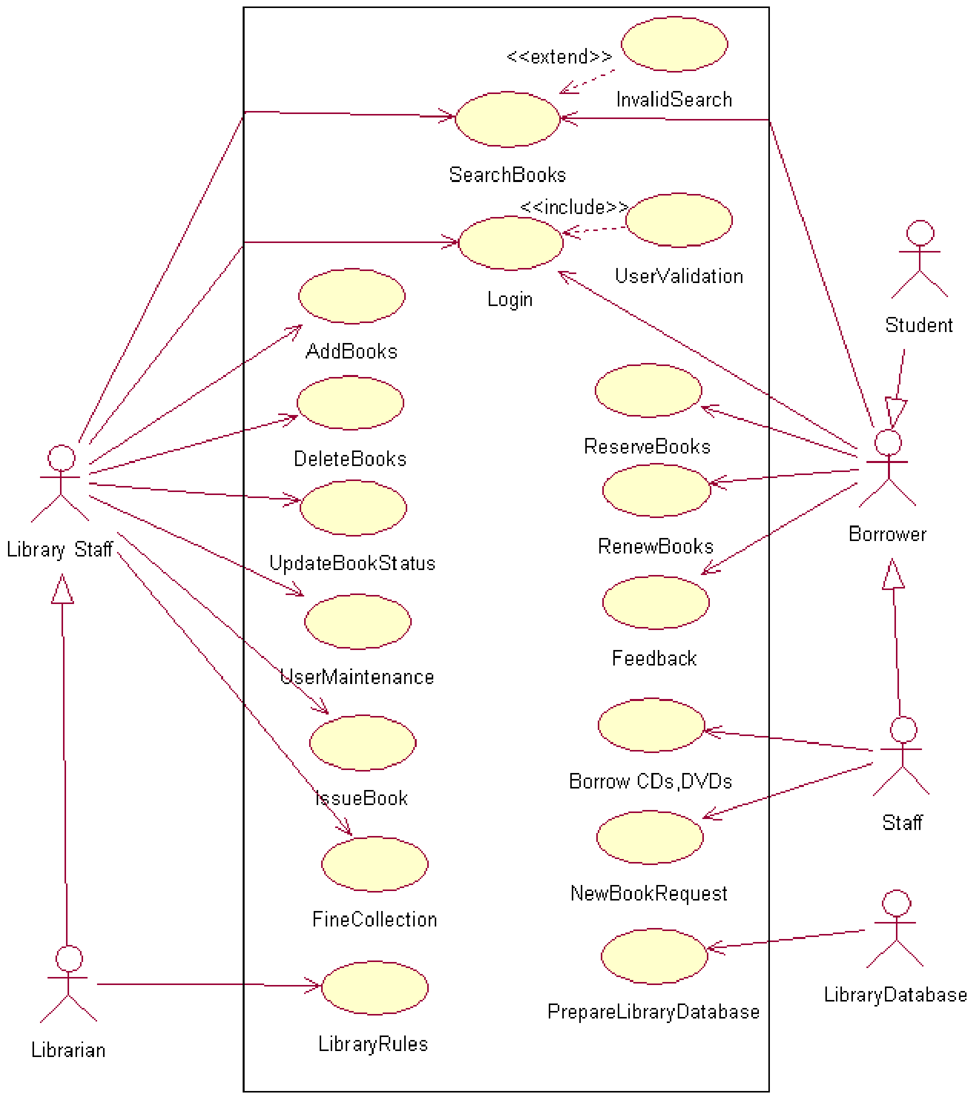


Fig: use case diagram

Aim: To draw sequence diagram for library application.

Software used: Rational Rose

Procedure: Create a new package from the database and name it as a sequence diagram from the toolbar, select the required tools and make corresponding connections.

Description: In this sequence diagram of use case view of the unified library application, we have four objects the member, library, librarian and title.

We can also represent the object lifeline and flow of control in sequence diagrams. Firstly a member enters into library and gives the details after which the librarian check for validity.

If validity check ok() evaluates true, the member searches for book. After the librarian issuing the book, the member collects it from library.

If the book is not found then the member can place order using the place order(). If the person returns the book late, then he should pay the fine and librarian collects it.

Result: We have successfully worked with the sequence diagram in usecase view for unified library application.

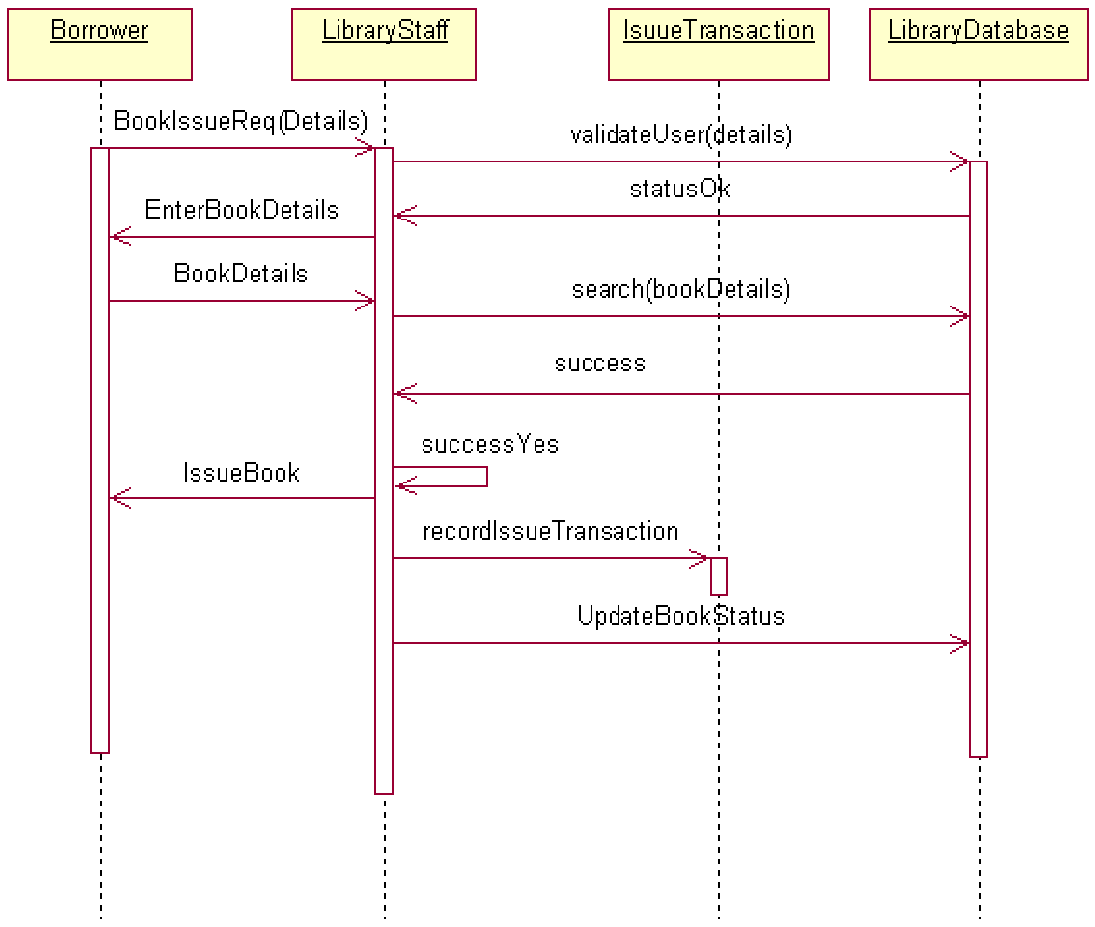


Fig: sequence diagram for issuing book

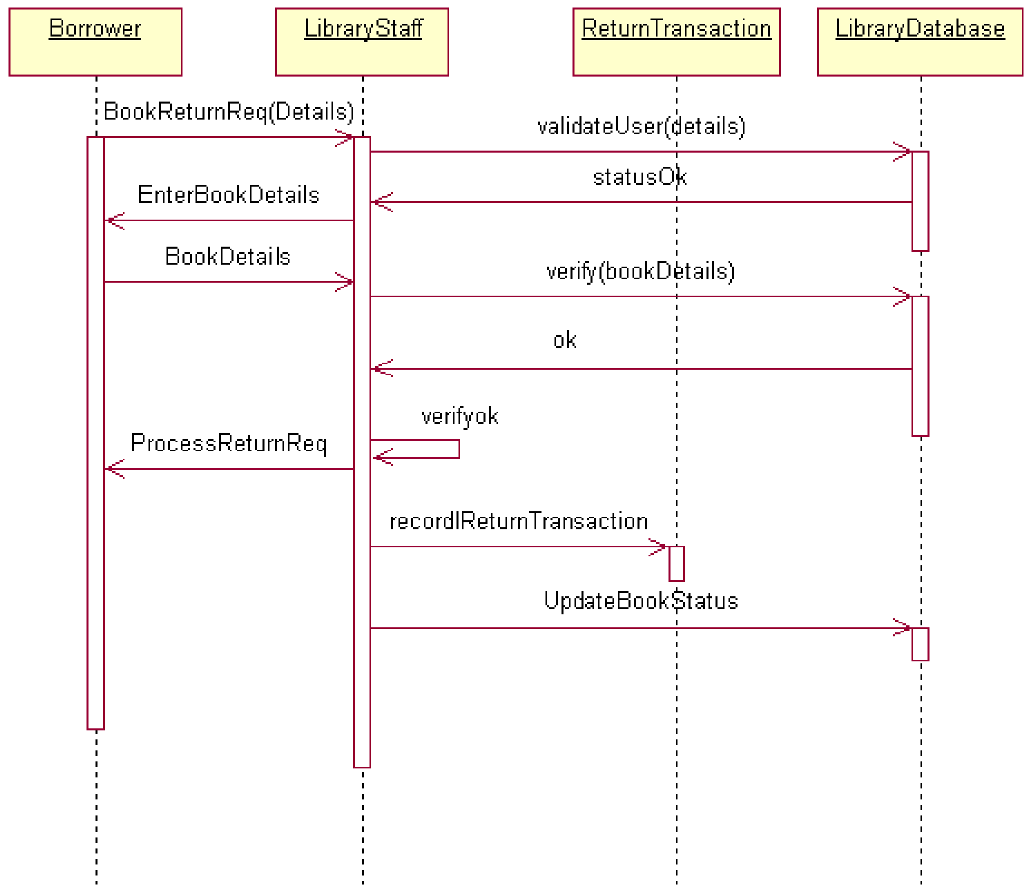


Fig: sequence diagram for return book

Aim : To draw collaboration diagram for library application.

Software used: Rational Rose

Procedure: Create a new package from the database and name it as a collaboration diagram From the toolbar, select the required tools and make corresponding connections.

Description: The collaboration diagram shows the structural organization of objects.

These are used to show complex interactions and multiple flow of control.

The object member can enter into library. Checks and collects the book and also return it back.

The member can also place order if he can't found the desired book. The object librarian can issues books and collects fine from the members.

It has also interaction such as validity check and placing reservations etc…

Result: We have successfully worked with the collaboration diagram in usecase view for unified library application.

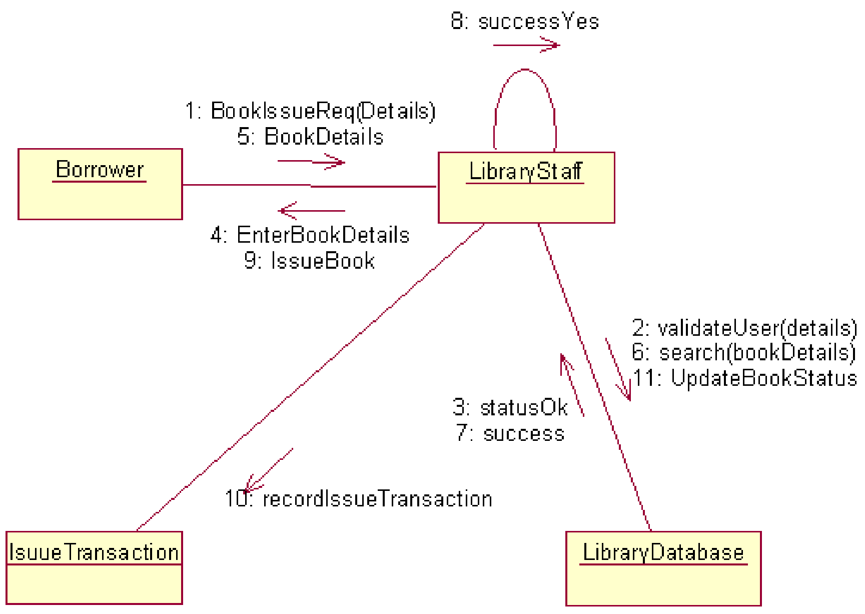


Fig: collaboration diagram for issuing book

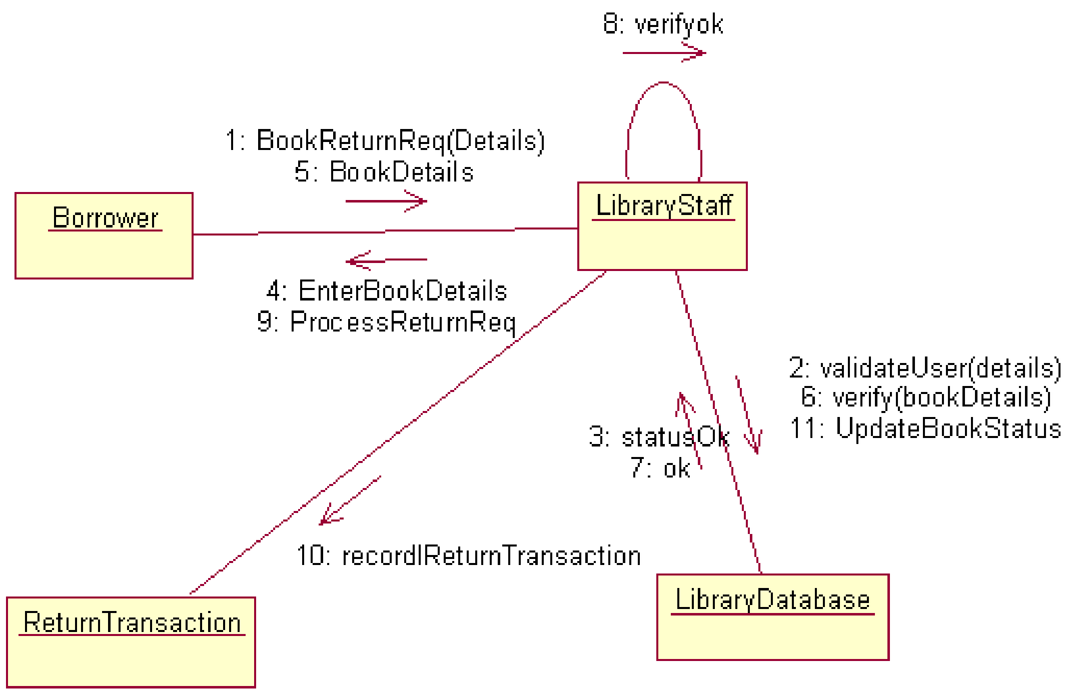


Fig: collaboration diagram for return book

Aim : To state chart diagram for library application.

Software used: Rational Rose

Procedure: Create a new package from the database and name it as a state chart diagram from the toolbar, select the required tools and make corresponding connections.

Description:

- Choose the context for state machine, whether it is a class, a use case, or the system as a whole.
- Choose the initial & final states of the objects.
- Decide on the stable states of the object by considering the conditions in which the object may exist for some identifiable period of time. Start with the high level states of the objects & only then consider its possible sub states.
- Decide on the meaningful partial ordering of stable states over the lifetime of the object.
- Decide on the events that may trigger a transition from state to state. Model these events as triggers to transitions that move from one legal ordering of states to another.
- Attach actions to these transitions and/or to these states.
- Consider ways to simplify your machine by using substrates, branches, forks, joins and history states.
- Check that all states are reachable under some combination of events.
- Check that no state is a dead from which no combination of events will transition the object out of that state.
- Trace through the state machine, either manually or by using tools, to check it against expected sequence of events & their responses.

Result: We have successfully worked with the collaboration diagram in usecase view for unified library application.

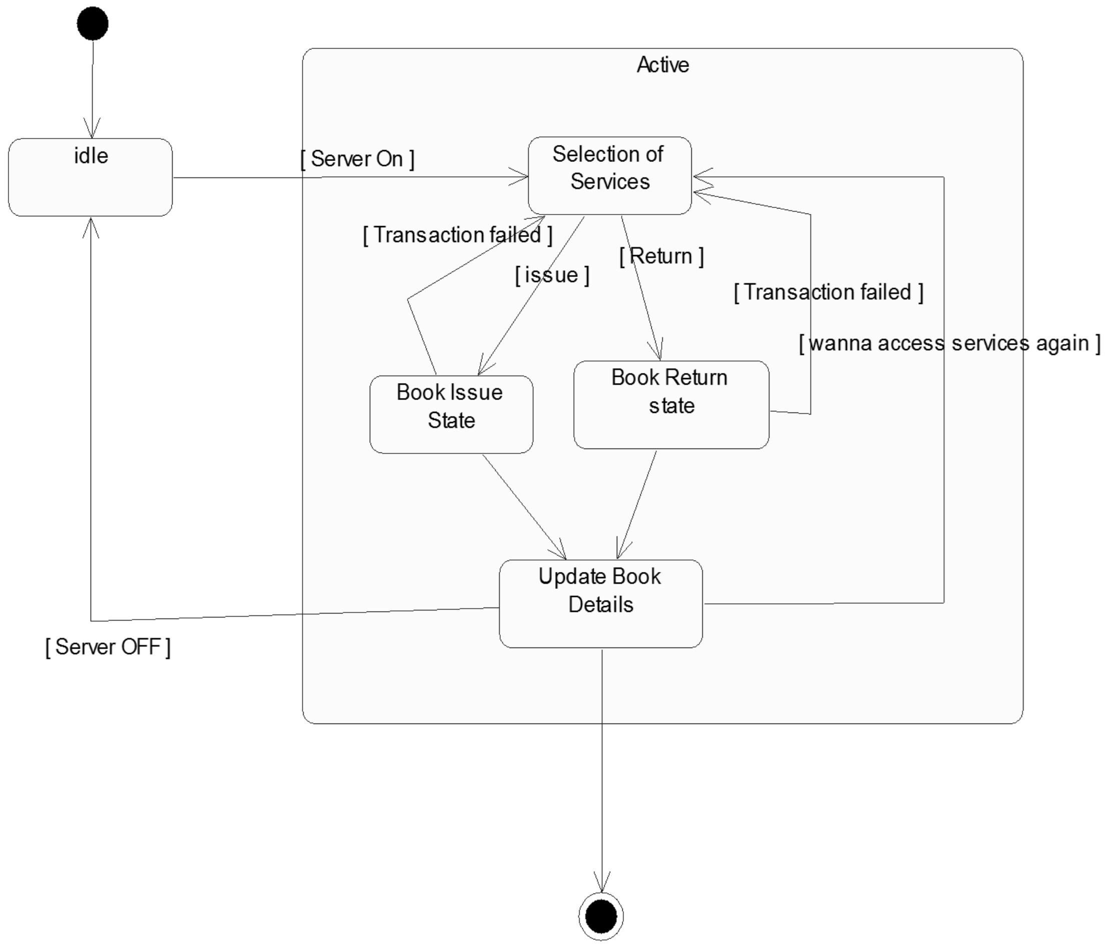


Fig: state chart diagram

Aim: To draw class diagram library application.

Software used: Rational Rose

Procedure:Create a new package from the database and name it as class diagram. From the toolbar, select the required tools and make corresponding connections.

Description: In modeling unified library application we have the classes such as library, title, librarian and barrower.

By using this class diagram we can easily understand the information of unified library application.

There is an aggregation relationship between library and librarian is a part of the library.A library can have one or more librarians which can be expressed in multiplicity as 1…*. There is an association relationship between title and barrower classes.

A title can have 0 or more barrowers. Thus the multiplicity is 0…*. A library can have any number of librarians as well as the barrowers.

There exists an association relation in between the barrower and the librarian classes where the barrower can talk with librarian.

The classes, books, and magazines are the child classes of title where there exists dependency relation in between them. Similarly the student and staff classes are the children of barrower classes.

Result: We have successfully worked with the class diagram in logical view for unified library application.

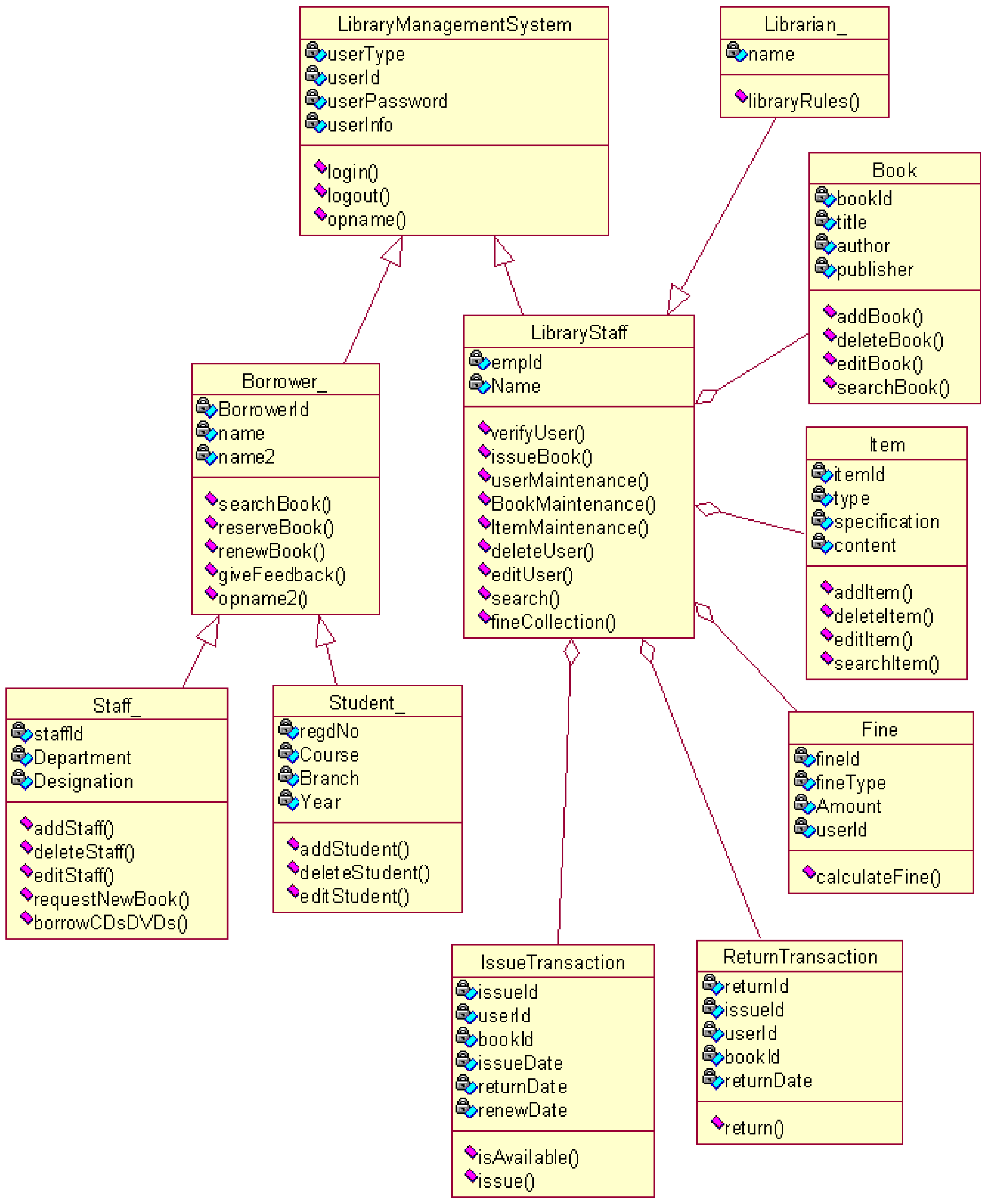


Fig: class diagram

]Aim : To component diagram for library application.

Software used: Rational Rose

Procedure: Create a new package from the database and name it as a component diagram from the toolbar, select the required tools and make corresponding connections.

Description:

- Identify the component libraries and executable files which are interacting with the system.
- Represent this executables and libraries as components.
- Show the relationships among all the components.
- Identify the files, tables, documents which are interacting with the system.
- Représenté files, tables, documents as components.
- Show the existing relationships among them generally dependency.
- Identify the seams in the model.
- Identify the interfaces which are interacting with the system.
- Set attributes and operation signatures for interfaces.
- Use either import or export relationship in b/w interfaces & components.
- Identify the source code which is interacting with the system.
- Set the version of the source code as a constraint to each source code.
- Represent source code as components.
- Show the relationships among components.
- Adorn with nodes, constraints and tag values.

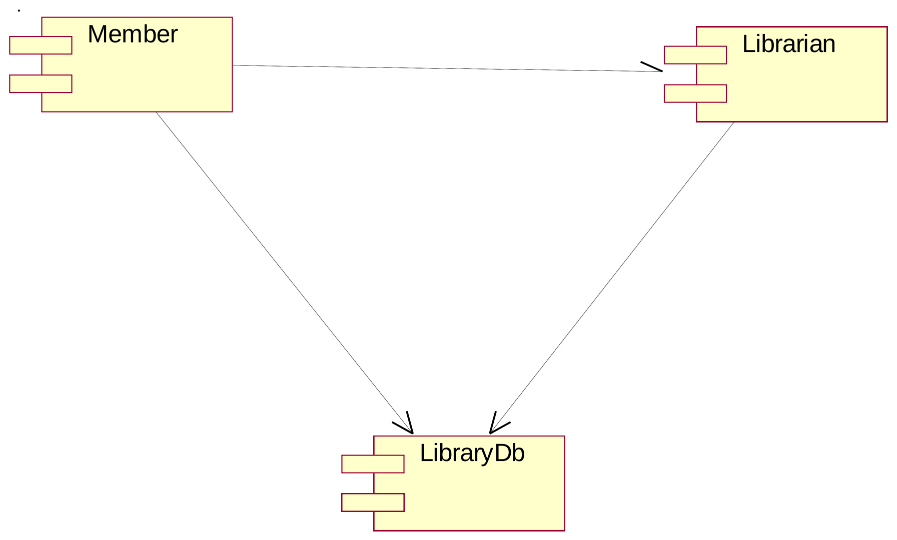


Fig: component diagram

Aim : To deployment diagram for library application.

Software used: Rational Rose

Procedure: Create a new package from the database and name it as a deployment diagram from the toolbar, select the required tools and make corresponding connections.

Description:

- Identify the processors which represent client & server.
- Provide the visual cue via stereotype classes.
- Group all the similar clients into one package.
- Provide the links among clients & servers.
- Provide the attributes & operations.
- Specify the components which are living on nodes.
- Adorn with nodes & constraints & draw the deployment diagram.

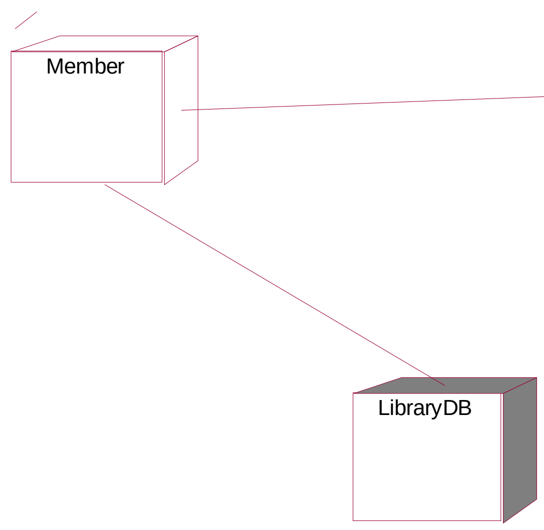


Fig: Deployment diagram

POINT OF SALE TERMINAL

Description:

A POS system is a computerized application used to record sales and handle payments. It is typically used in a retail store. It includes hardware components such as computer and barcodes scanner and software to run the system. The systems must be relatively fault tolerant i; e even if the remote services are temporarily unavailable, they must still be capable of capturing sales and handling cash payments. A POS system must support multiple client side terminals and interfaces such as browsers, PDA {Personal Digital assistance} touch screens.

This case study demonstrates the iterative incremental process covering the analysis and incremental process.

a) Identify and analyze events

- cashier
    - login
    - cash out
    - receipt generation
    - scan item
- Customer
    - Purchase Item
    - Select Item
    - Payment by cash
    - Refund item
- Manager
    - List of all items purchased
    - Stock of available Items

    - Add new Items
    - Delete Items
    - Manage sales-view sales details
- Check out
    - Scan item
    - Calculate tax
    - Payment
- Product Cat log
    - Show product
    - Add new Product

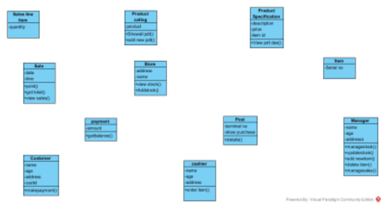
b) Identify Use cases

- Enter Item
- By item
- Get balance
- Manage stock
- Update stock
- Delete item
- View sales details
- View product description
- Make payment
- Quantity

c) Develop event table

	Customer	Sales	Product	Database
Item	*		*	
Payment	*			
Buy item	*	*	*	*
Status of Stock		*	*	*
Product check out			*	
Customer register	+	*		*
Customer login	+			

d) Identify & analyze domain classes



e) Develop CRUD matrix to represent relationships between use cases and problem domain classes

Data entity	CRUD	Resulting user case
Customer	Create Read/Report Update Delete	Add new customer Find customer Generate customer Purchase list Update customer information Delete inactive customer
Inventory item	Create Read/Report Update Delete	Add new inventory item View item Generate item list Update Item list Delete Item list
Sales	Create Read/Report Update Delete	Add new Sales item View sales View sales item Update sales Item Delete sales item

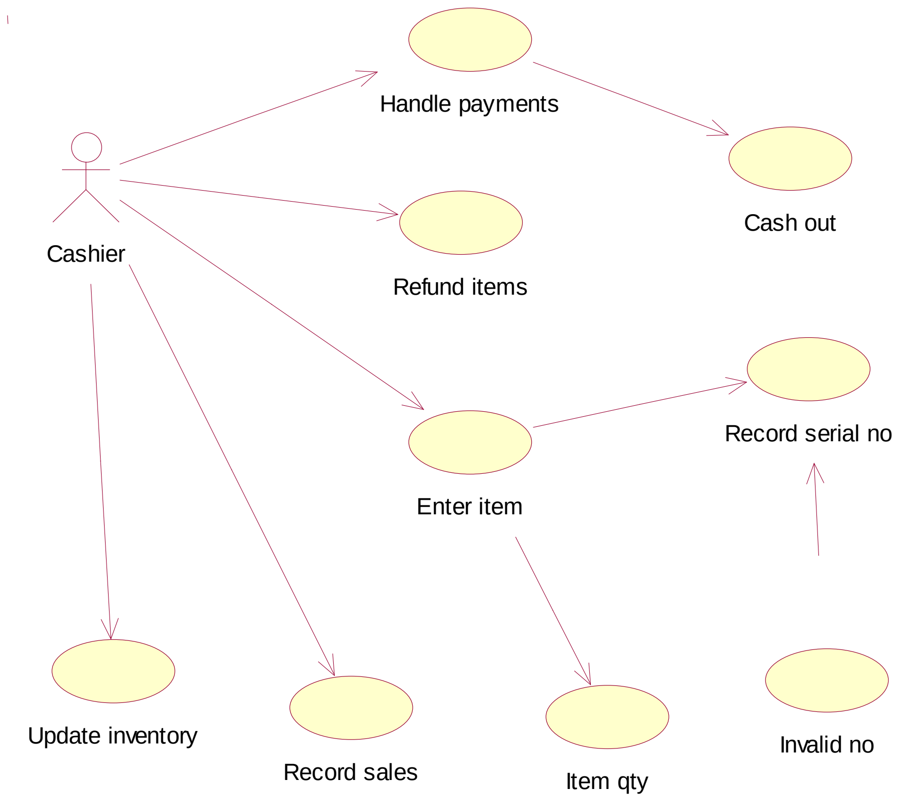


Fig: Use case diagram

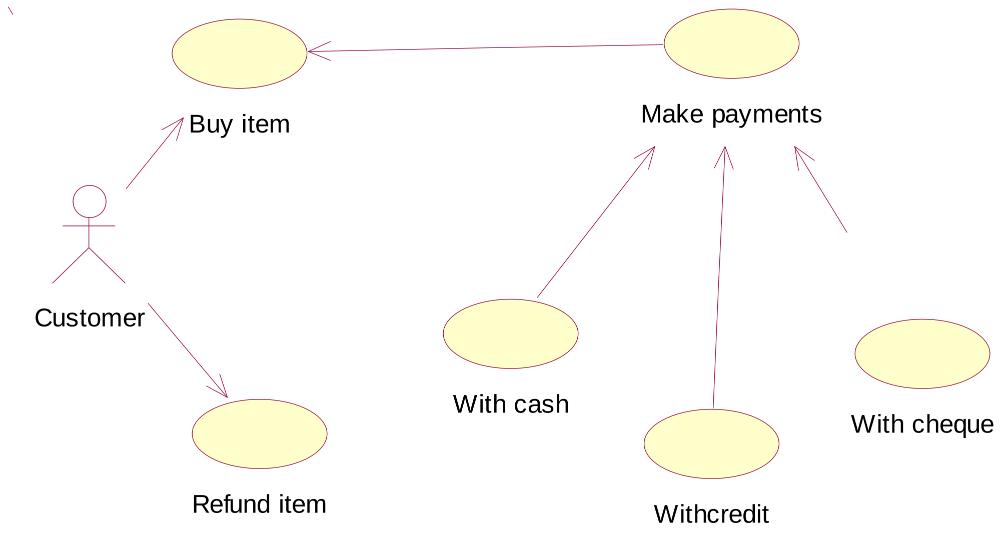


Fig: use case diagramfor action customer

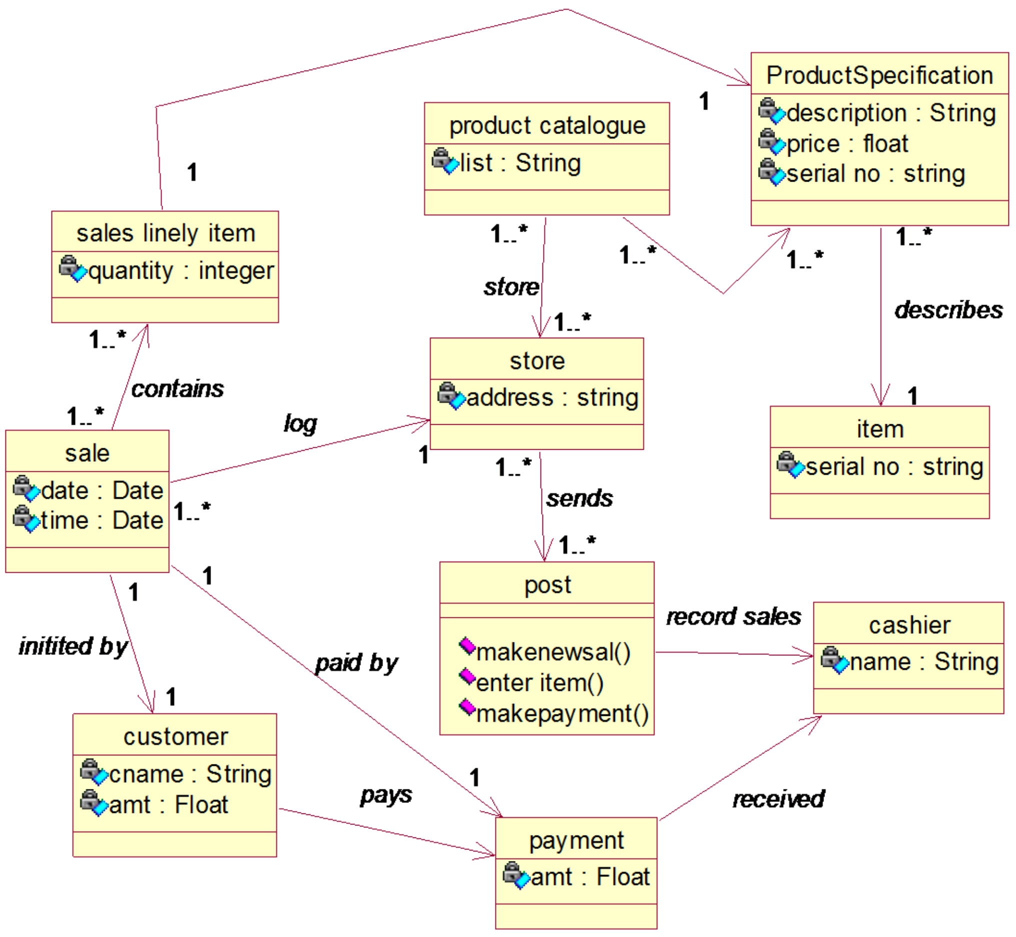


Fig: Class diagram for POS

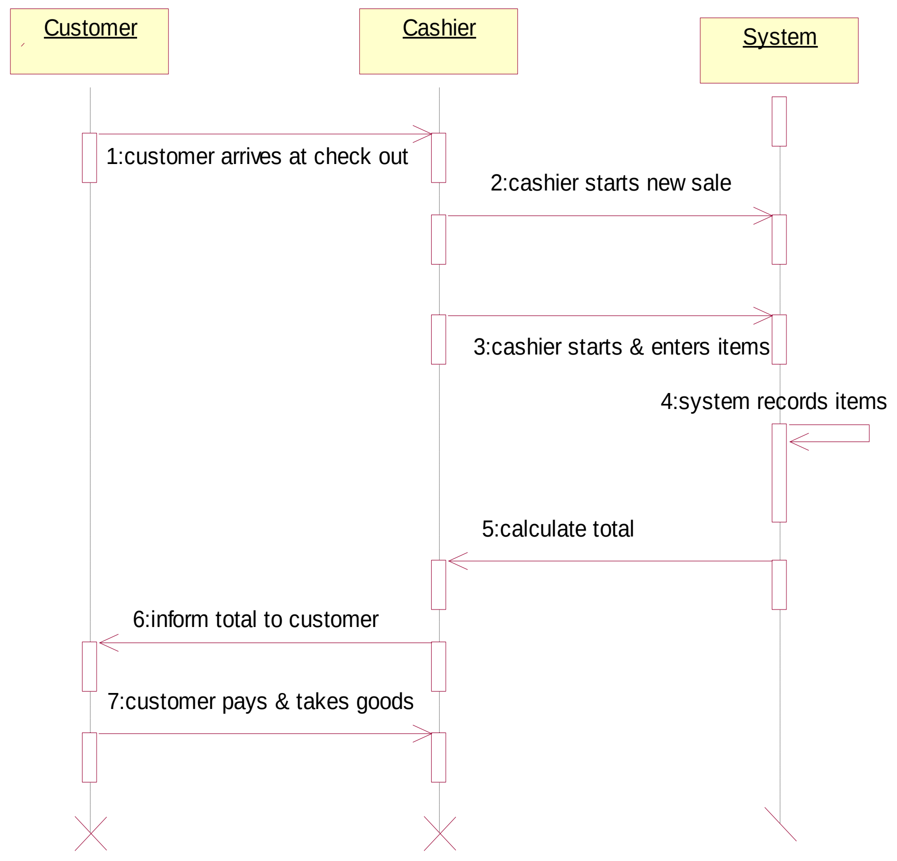

Customer
Cashier
System
1:customer arrives at check out
2:cashier starts new sale
3:cashier starts & enters items
4:system records items
5:calculate total
6:inform total to customer
7:customer pays & takes goods

Fig: Sequence diagram for POS

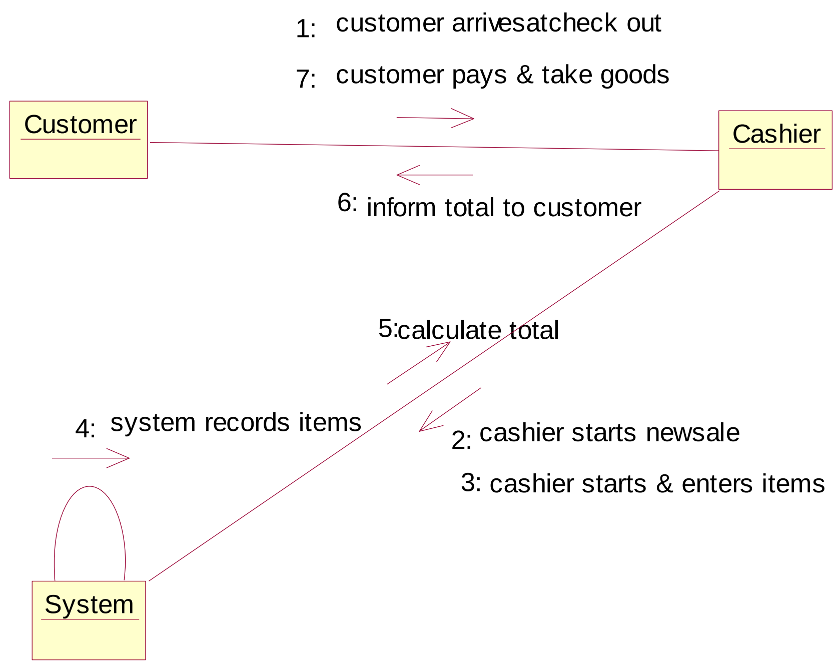

1: customer arrivesatcheck out
7: customer pays & take goods
Customer
Cashier
6: inform total to customer
5:calculate total
4: system records items
2: cashier starts newsale
3: cashier starts & enters items
System

Fig: collaboration diagram for POS

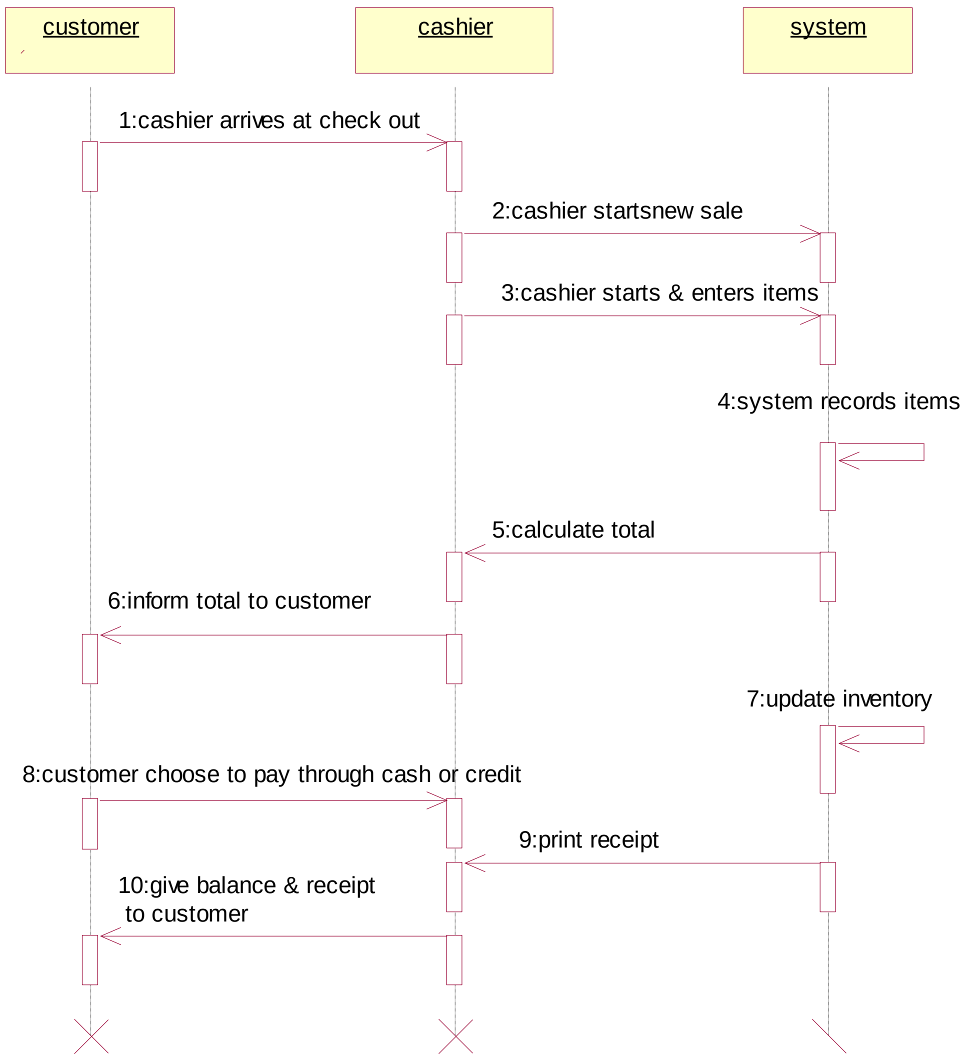

customer
cashier
system
1:cashier arrives at check out
2:cashier startsnew sale
3:cashier starts & enters items
4:system records items
5:calculate total
6:inform total to customer
7:update inventory
8:customer choose to pay through cash or credit
9:print receipt
10:give balance & receipt
to customer

Fig: Sequence diagram for buy items cash/credit

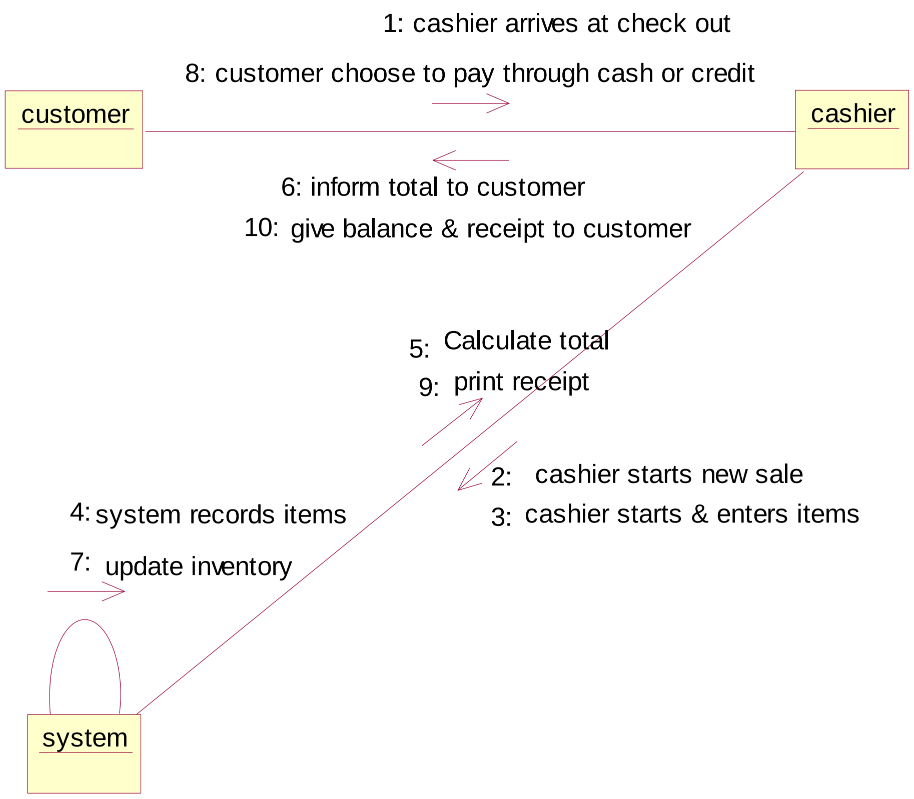

1: cashier arrives at check out
8: customer choose to pay through cash or credit
customer
cashier
6: inform total to customer
10: give balance & receipt to customer
5: Calculate total
9: print receipt
2: cashier starts new sale
3: cashier starts & enters items
4: system records items
7: update inventory
system

Fig: collaboration diagram for buy items cash/credit

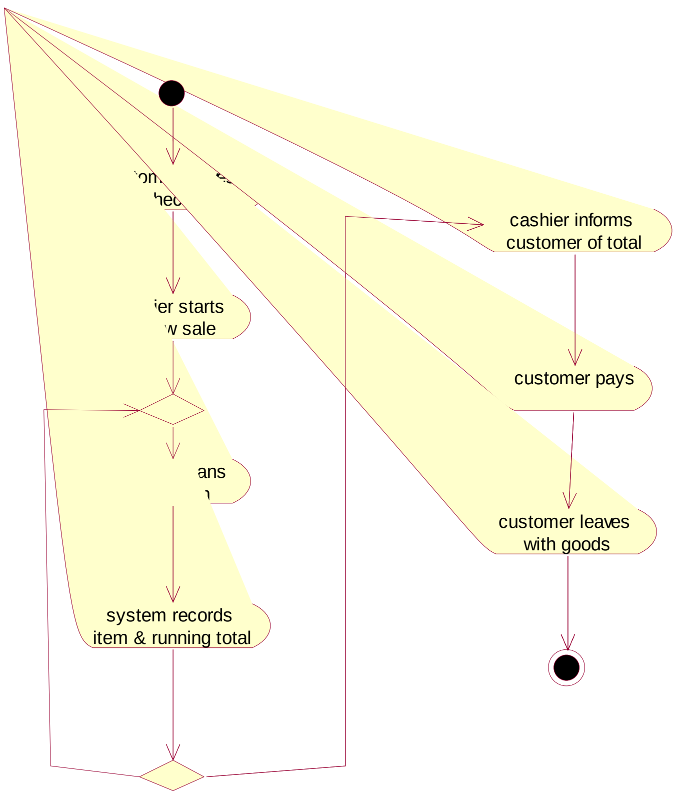


Fig: activity diagram for POS

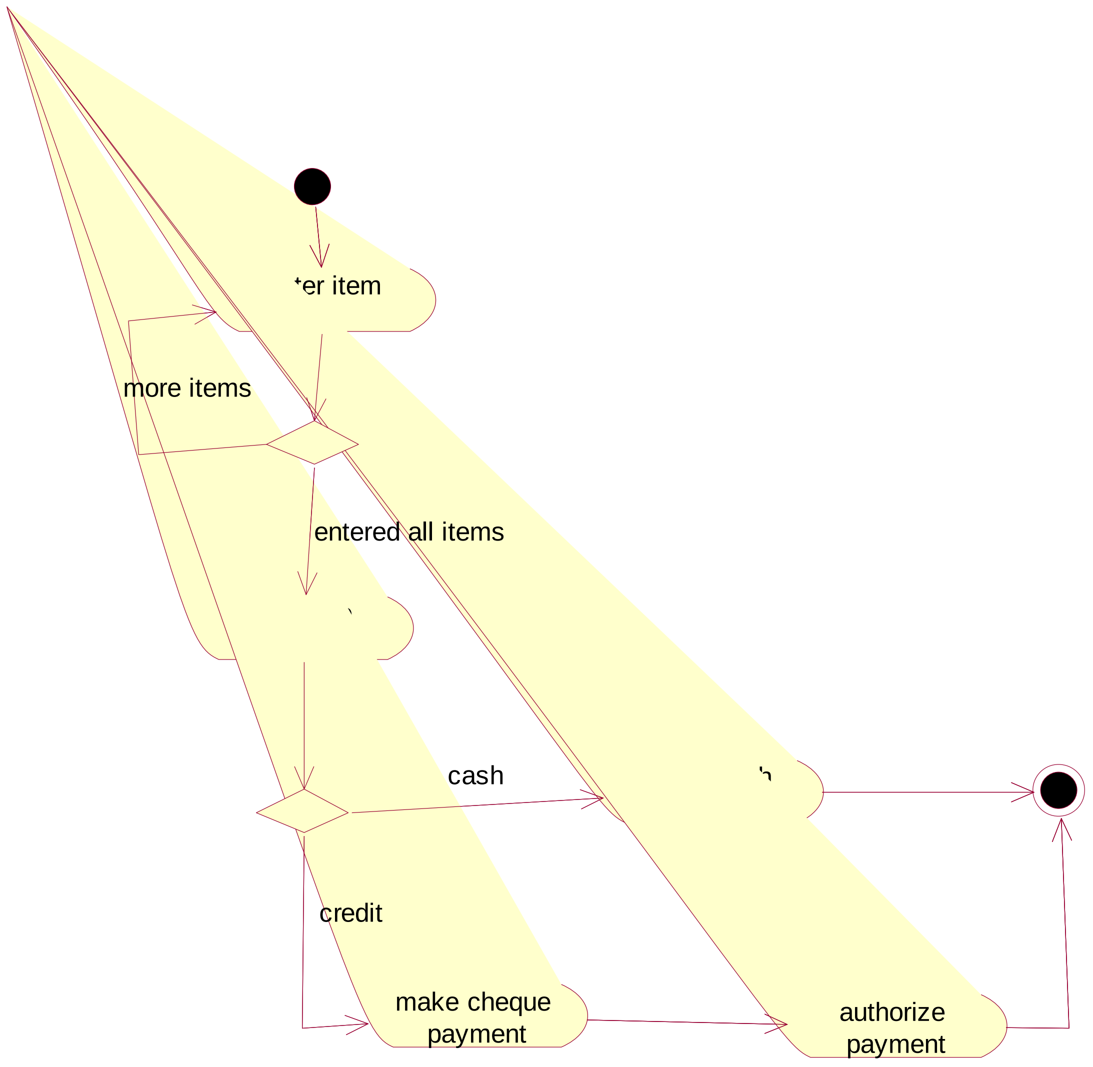


Fig: activity diagram to buy items

GRASP PATTERNS:

What are Patterns?

• **Pattern**

– Is a named and well-known problem/solution pairthat can be applied in new contexts

• With advice on how to apply it in novel situations and discussion of its trade-offs, implementations, variations.

Pattern Name: Information Expert

Problem: What is a basic principle by which to assignresponsibilities to objects?

Solution: Assign a responsibility to the class that has theinformation needed to fulfill it.

– Naming a pattern, design idea, or principle has thefollowing advantages:

• It supports chunking and incorporating that concept into ourunderstanding and memory.

• It facilitates communication

New pattern should be considered an oxymoron if itdescribes a new idea.

– The very term "pattern" suggests a long-repeating thing.

– The point of design patterns is not to express new design ideas.

Quite the opposite — great patterns attempt to codify existingtried-and-true knowledge, idioms, and principles

• The Gang-of-Four Design Patterns.

– 23 patterns (Strategy, Adaptor, …)

GRASP (principles or patterns)– General Responsibility Assignment SoftwarePatterns

Nine GRASP Principles:

• Low Coupling

• High Cohesion

• Information Expert

• Creator

• Controller

• Polymorphism

• Indirection

• Pure Fabrication

• Protected Variations

Controller

The **Controller** pattern assigns the responsibility of dealing with system events to a non-UI class that represents the overall system or a use case scenario. A Controller object is a non-user interface object responsible for receiving or handling a system event.

A use case controller should be used to deal with *all* system events of a use case, and may be used for more than one use case (for instance, for use cases *Create User* and*Delete User*, one can have a single *UserController*, instead of two separate use case controllers).

It is defined as the first object beyond the UI layer that receives and coordinates ("controls") a system operation. The controller should delegate the work that needs to be done to other objects; it coordinates or controls the activity. It should not do much work itself. The GRASP Controller can be thought of as being a part of the Application/Service layer (assuming that the application has made an explicit distinction between the application/service layer and the domain layer) in an object-oriented system with Common layers in an information system logical architecture.

Creator

Creation of objects is one of the most common activities in an object-oriented system. Which class is responsible for creating objects is a fundamental property of the relationship between objects of particular classes.

In general, a class `B` should be responsible for creating instances of class `A` if one, or preferably more, of the following apply:

- Instances of `B` contain or compositely aggregate instances of `A`
- Instances of `B` record instances of `A`
- Instances of `B` closely use instances of `A`
- Instances of `B` have the initializing information for instances of `A` and pass it on creation.

High cohesion

High cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of low coupling. High cohesion means that the responsibilities of a given element are strongly related and highly focused. Breaking programs into classes and subsystems is an example of activities that increase the cohesive properties of a system. Alternatively, low cohesion is a situation in which a given element has too many unrelated responsibilities. Elements with low cohesion often suffer from being hard to comprehend, hard to reuse, hard to maintain and averse to change.

Indirection

The **indirection** pattern supports low coupling (and reuse potential) between two elements by assigning the responsibility of mediation between them to an intermediate object. An example of this is the introduction of a controller component for mediation between data (model) and its representation (view) in the Model-view-controller pattern.

Information expert

Information expert (also **expert** or the **expert principle**) is a principle used to determine where to delegate responsibilities. These responsibilities include methods, computed fields, and so on.

Using the principle of information expert, a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfill it, and then determine where that information is stored.

Information Expert will lead to placing the responsibility on the class with the most information required to fulfill it.

Low coupling

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. **Low coupling** is an evaluative pattern that dictates how to assign responsibilities to support:

- lower dependency between the classes,
- change in one class having lower impact on other classes,
- higher reuse potential.

Polymorphism

According to **polymorphism**, responsibility of defining the variation of behaviors based on type is assigned to the type for which this variation happens. This is achieved usingpolymorphic operations. The user of the type should use polymorphic operations instead of explicit branching based on type.

Protected variations

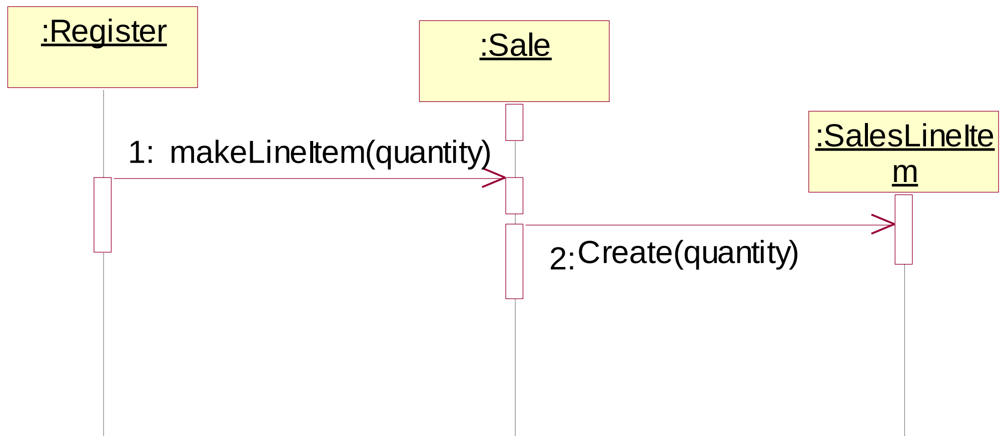
The **protected variations** pattern protects elements from the variations on other elements (objects, systems, subsystems) by wrapping the focus of instability with an interfaceand using polymorphism to create various implementations of this interface.

Pure fabrication

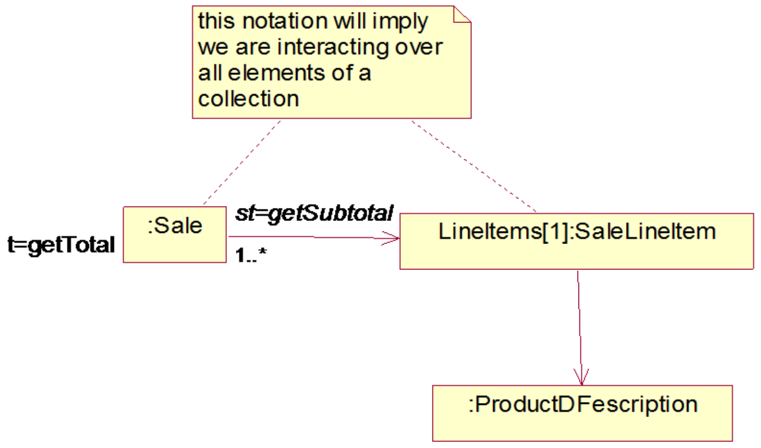
A **pure fabrication** is a class that does not represent a concept in the problem domain, specially made up to achieve low coupling, high cohesion, and the reuse potential thereof derived (when a solution presented by the *information expert* pattern does not). This kind of class is called a "service" in domain-driven design.

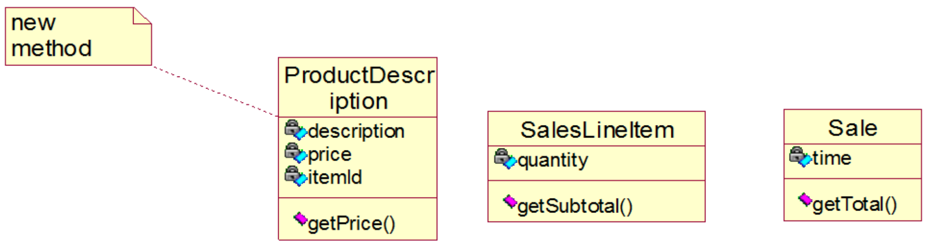
GRASP PATTERNS FOR POS:

1: Creator

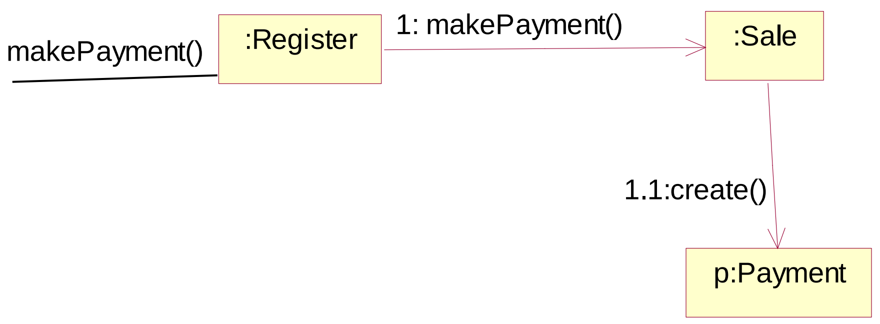


2: Information Expert

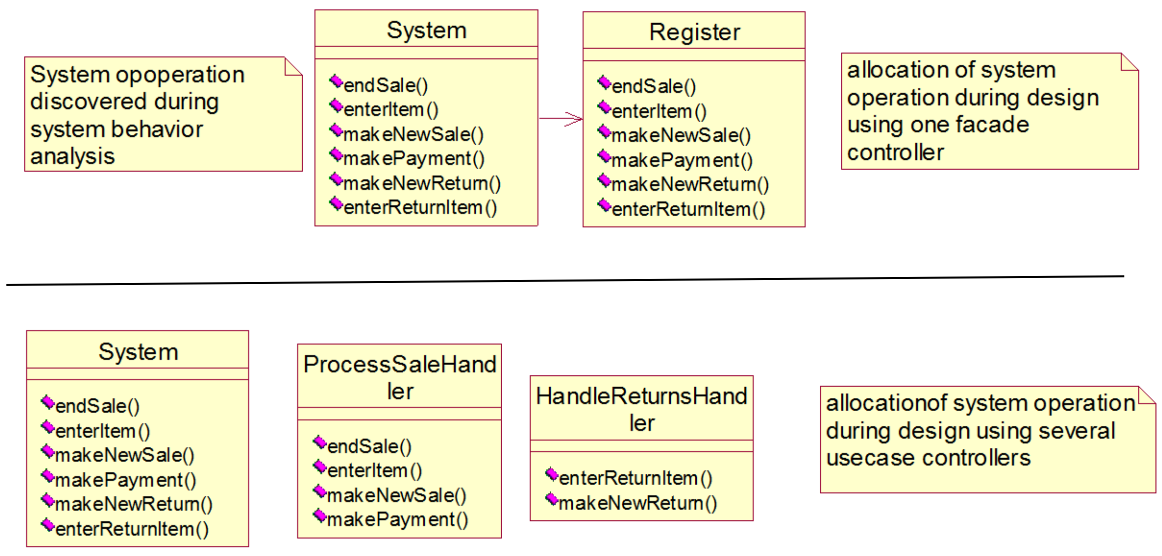

this notation will imply we are interacting over all elements of a collection
t=getTotal
:Sale
st=getSubtotal
1..*
LineItems[1]:SaleLineItem
:ProductDFescription



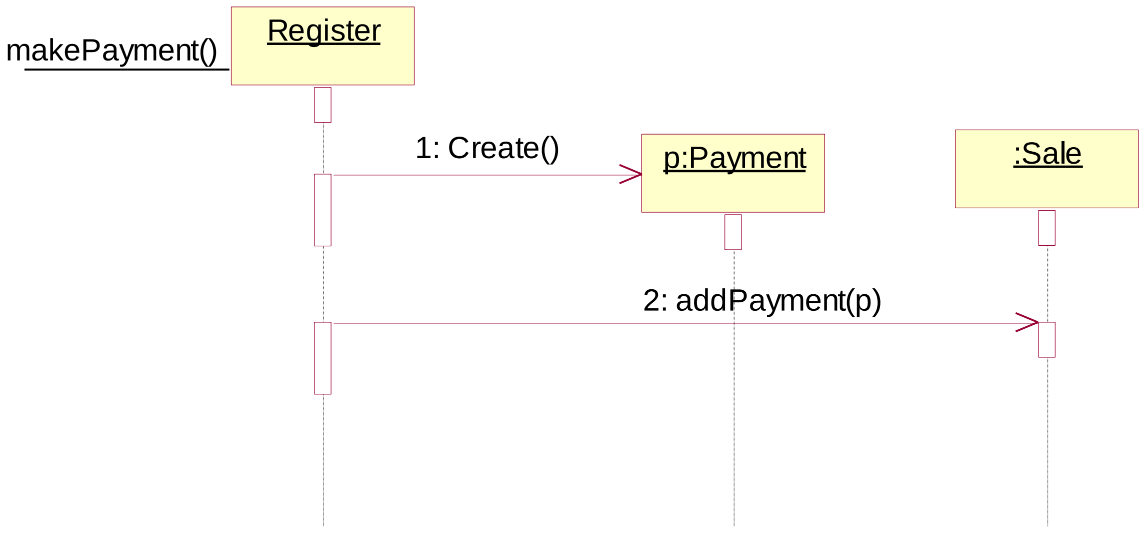
3: Low coupling



4: Controller



5: High cohesion



ONLINE BOOK SHOP

AIM: To create a UML diagram of ONLINE BOOK SHOP.

DESCRIPTION:

In the ONLINE BOOK SHOP, the User may Login to any website containing Book Shopping, and can perform the following operations: Search for a particular Book, selecting a book, adding a Book to a cart, Updating Cart, Purchase Book, Giving Feedback, Checking out Payment details etc. Whereas the Admin will be there Controlling the Operations performed by the User. The Operations done by the Admin are: Adding new Items to the Cart, Modifying Customer Order, and Checking Availability etc. The Bank Server will be connected to the Website to Check the Payment Details.

Diagrams

1. USECASE Diagram

Actors

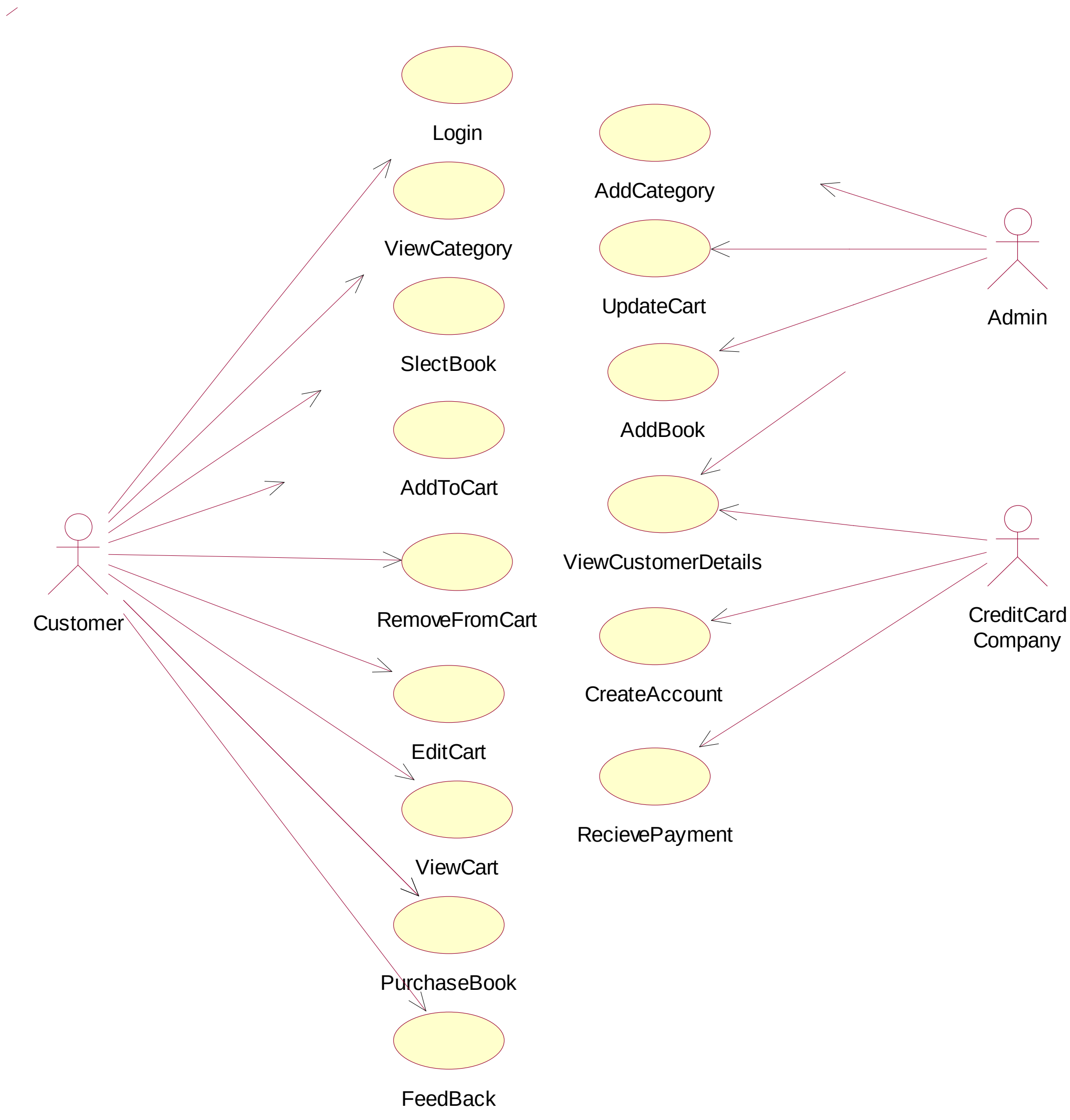
- User
- Admin
- Visitor
- Credit Card Company

Use cases

- Logging into the website, Selecting book, Adding Book to the Cart
- Updating the Cart, Giving feedback, Payment
- Updating Items in the website, Verifying Customer details
- Creating Accounts, Deleting Accounts, Receiving amount
- Sending Notification to Admin etc.

Use case diagram gives a graphical overview of the actors involved in a system, different functions needed by those actors and how these different functions are interacted.

Use case diagrams are the most known type of the Behavioral UML diagrams.

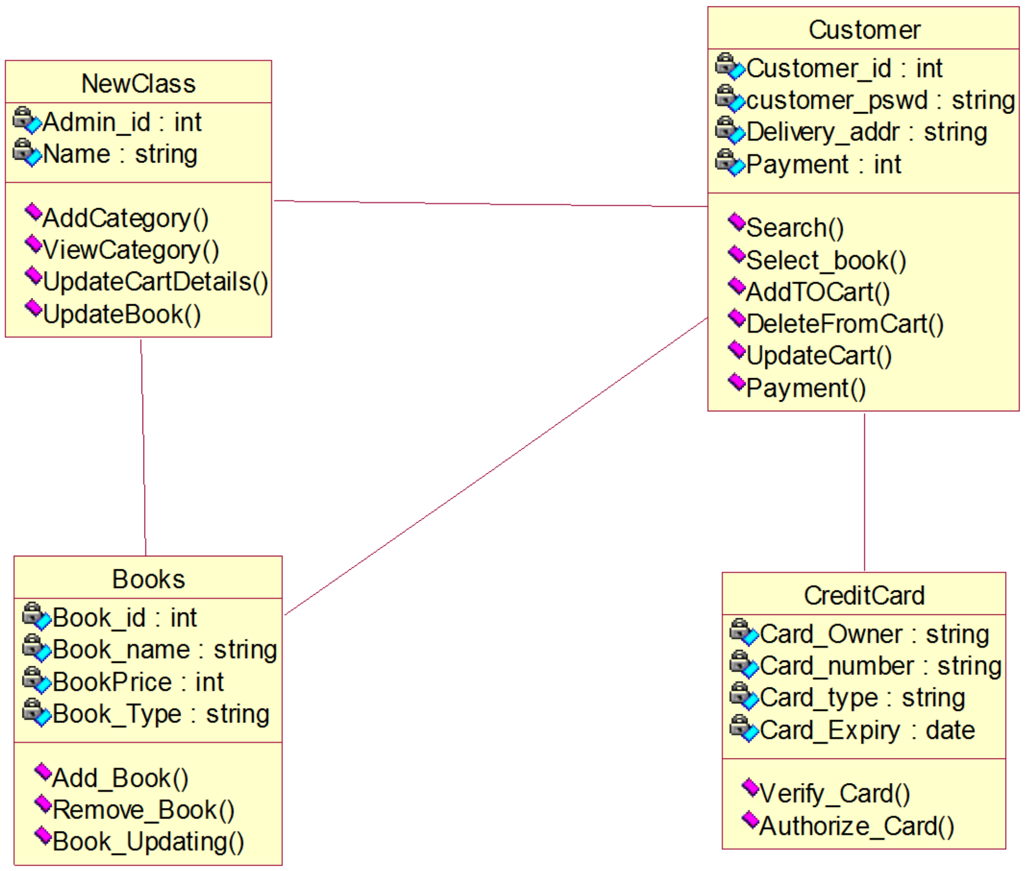


2. Class Diagram

The identified Classes are

- Customer
- Admin
- Books
- Credit Card

Class diagrams are arguably the most used UML diagram type. It is the main building block of any Object Oriented solution. It shows the classes in a system, attributes and operations of each class and the relationship between each class.



3. Component Diagram

The Components Are

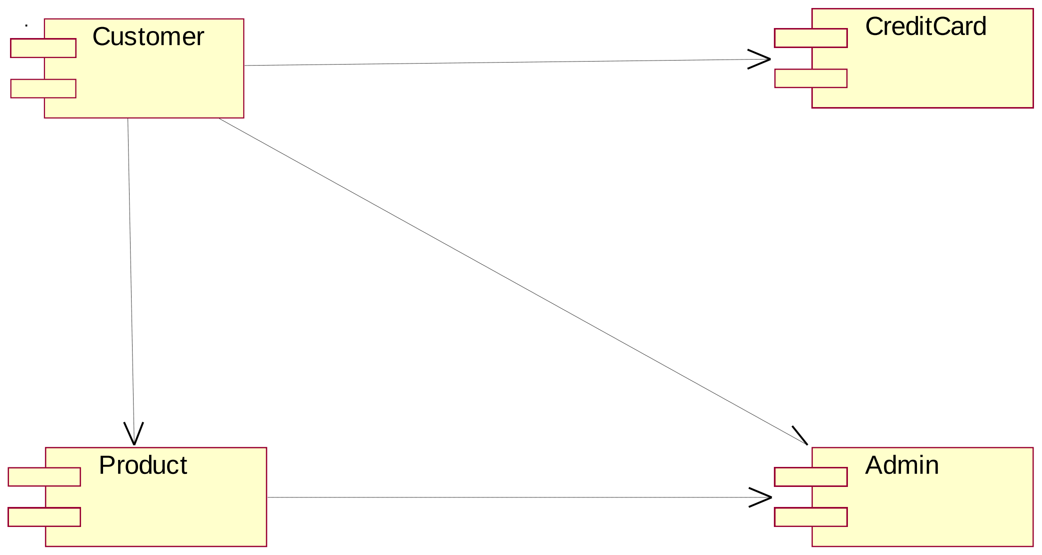
- Customer
- Admin
- Books
- Credit Card

A component Diagram displays the Structural relationship of components of a software system.

These are mostly used when working with complex systems that have many components. Components communicate with each other using Interfaces. The interfaces are liked using connectors.

A component diagram represents a modular part of a system that encapsulates its contents.

The component diagram can show components, dependencies between components and the way in which they are connected to.



4. Deployment Diagram

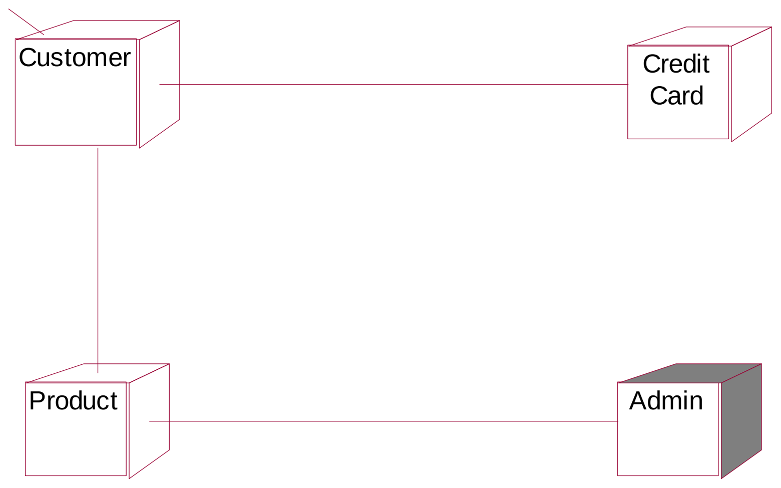
The Nodes in the Deployment Diagram are

- Customer
- Admin
- Books
- Credit Card

A Deployment Diagram shows the hardware of your system and the software in that hardware. Deployment Diagrams are useful when your software solution is deployed across multiple machines with each having a unique configuration.

The Deployment diagrams are represented using nodes. There are two types of nodes

- Device Node
- Processer Node

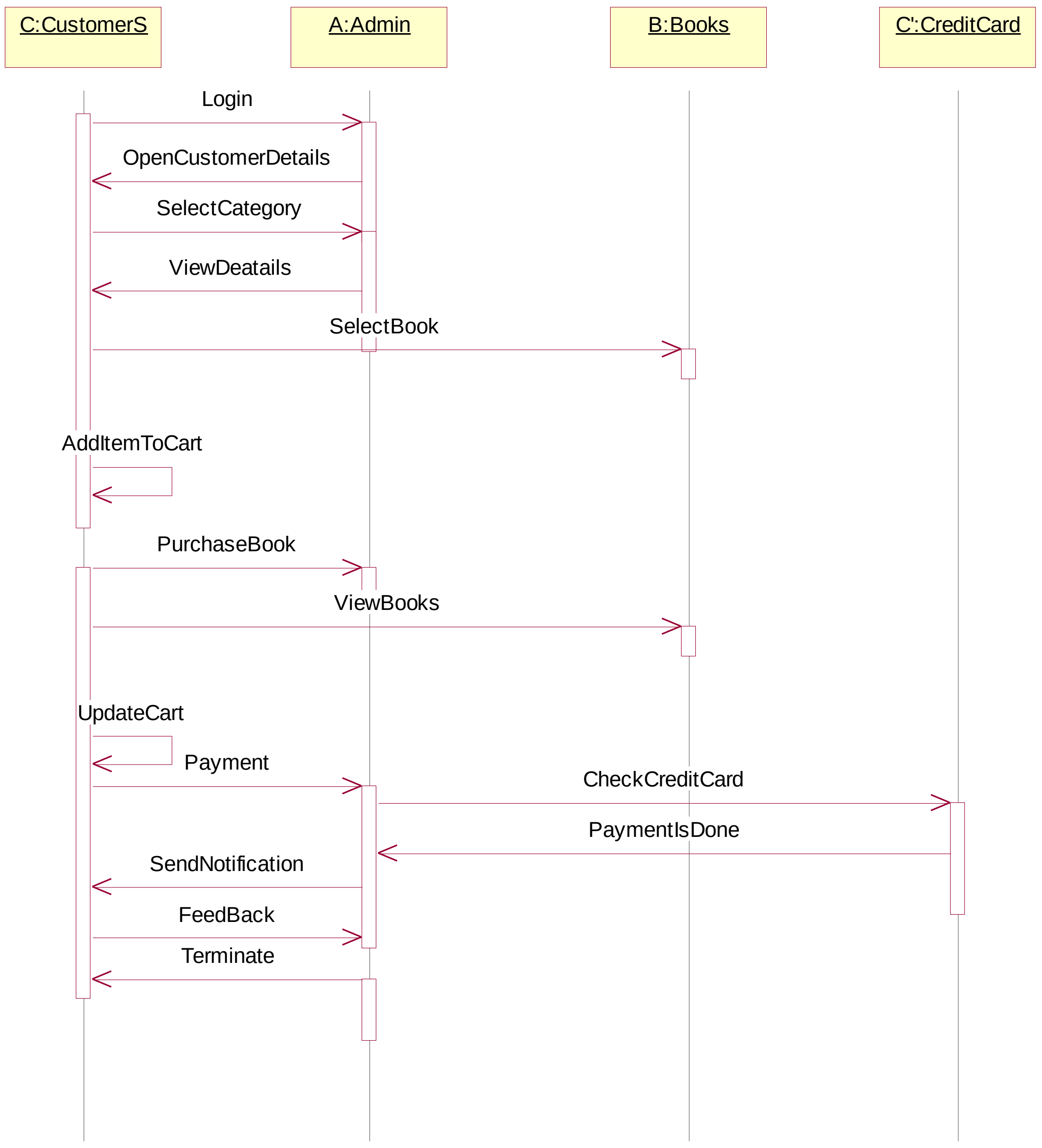


5. Sequence Diagram

The objects in the Sequence Diagram are

- Customer object
- Books object
- Admin object
- Credit Card object

Sequence Diagrams in UML shows how object interact with each other and the order those interactions occur. It is important to note that they show the interactions for a particular scenario. The Processes are represented vertically and interactions are shown as arrows.

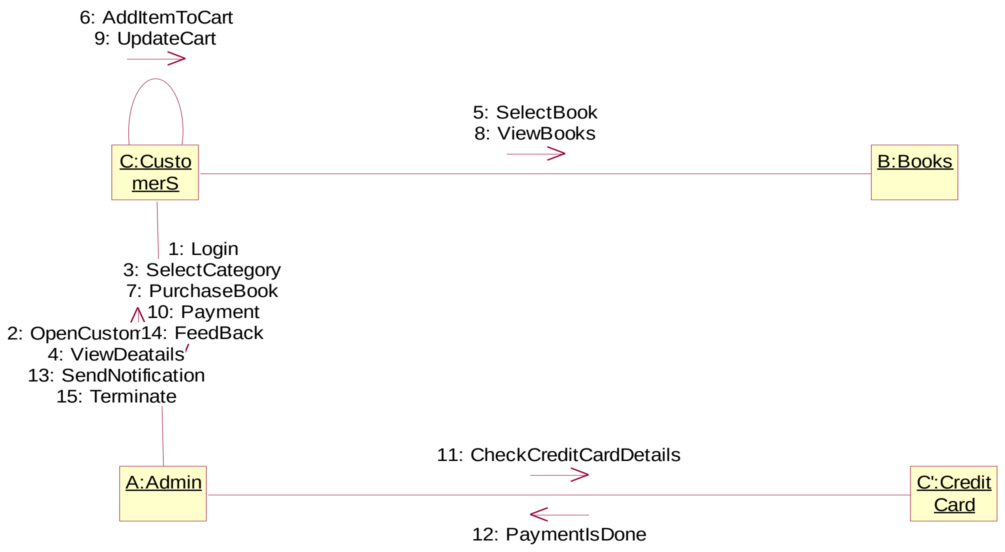


6. Collaboration Diagram

The objects in the Collaboration diagram are same as of Sequence diagram. They are

- Customer object
- Books object
- Admin object
- Credit Card object

Collaboration Diagrams are very similar to Activity Diagrams. While activity diagrams show a sequence of processes, Collaboration overview sequence of diagrams. It is similar to sequence diagrams but the focus is on messages passed between objects. The same information can be represented using a sequence diagram and different objects.
The Collaboration diagram describes interactions among objects in terms of sequenced messages; these are also called as Communication Diagrams or Interaction Diagrams.



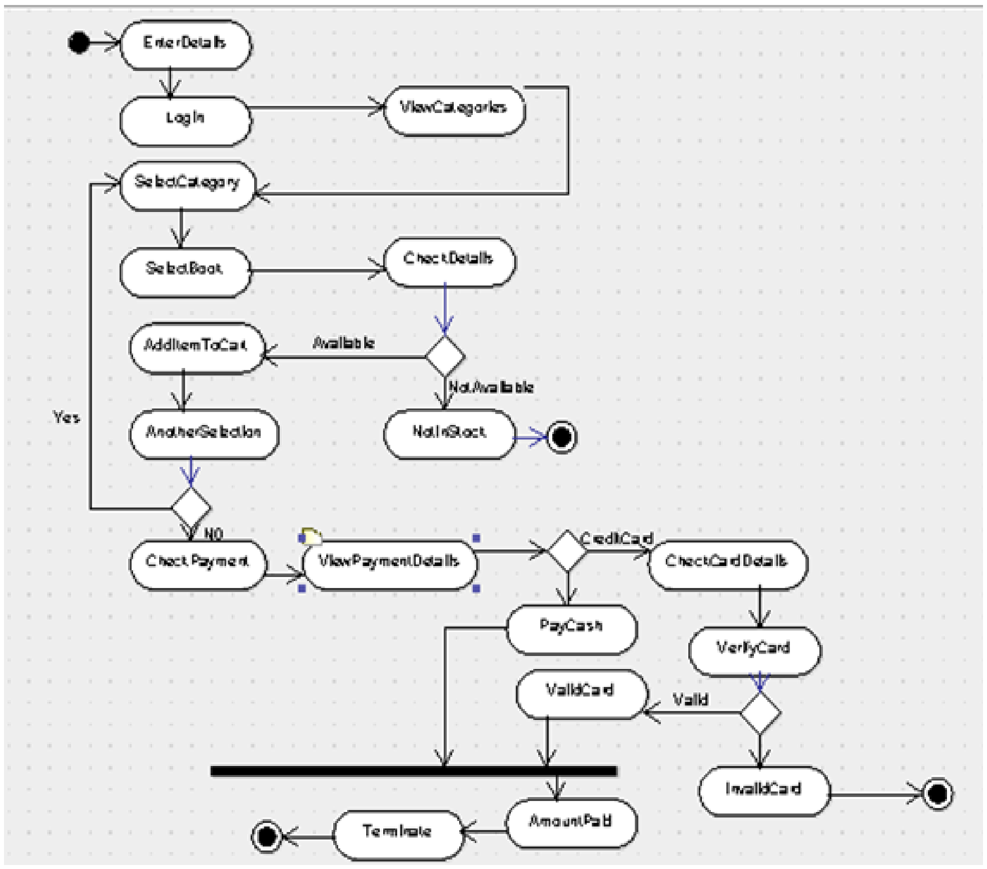
7. Activity Diagram

The objects in the Activity Diagram are

- Customer object
- Books object
- Admin object
- Credit Card object

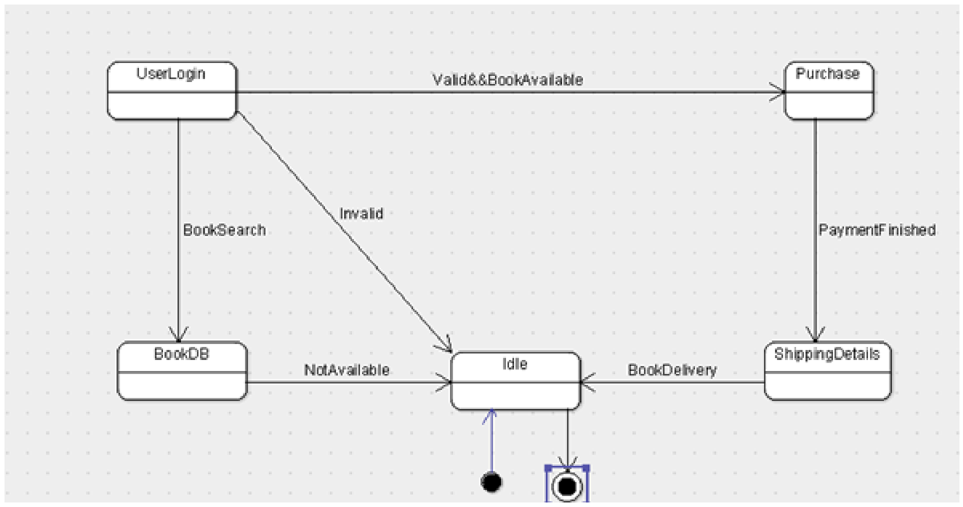
Activity Diagrams represent workflows in a graphical way. They can be used to describe business workflow or the operational workflow of any component in a system. Sometimes activity diagrams are used as an alternative to State machine Diagrams.

The Activity Diagrams are used to display the sequence of activities. They are nothing but the graphical representations of workflows of stepwise activities and actions.



8. State Chart Diagram

State machine diagrams are similar to Activity Diagrams although notations and usage changes a bit. They are sometimes known as state diagrams or State Chart Diagrams as well. They are very useful to describe the Behavior of objects that act different according to the state they are at the moment. They are most commonly used to model the dynamic behavior of classes.



Customer Support System

Take an Example of Online Shopping

a) Identify and analyze events

- View Items
  - -Search Items
  - -Browse Items
  - -View Recommended Items
  - -Add Shopping cart
  - -Add Wish List
- Check Out
  - -Customer Authentication
  - -View/Update Shopping cart
  - -Calculate Taxes and shipping
  - -Payment
- Customer Authentication
  - -Usersign-In
  - -Remember Me

b) Identify Use cases

- View Items
- Check Out
- Make Purchase
- Client Register
- Login
- Logout
- View order status
- Request order

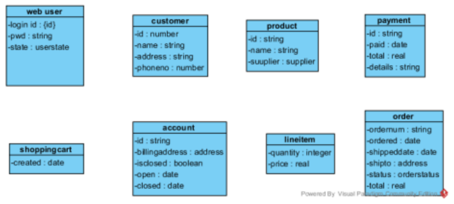
- Finish Order
- Payment
- Order cancellation
- Change account information
- Search Products

c) Develop event table

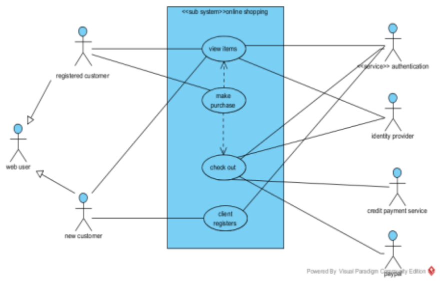
	Customer	Order	Product	Database
Customer registered	+			
Unsubscribed Customer	+			
Product Add to cart	*	*	*	
Product Remove from cart	*	*	*	
Place Order	*	+	*	*
Finish Order		*		
Product registration			+	*
Order Processing		*	*	*
Product Shipment			+	*
Product Delivery				*

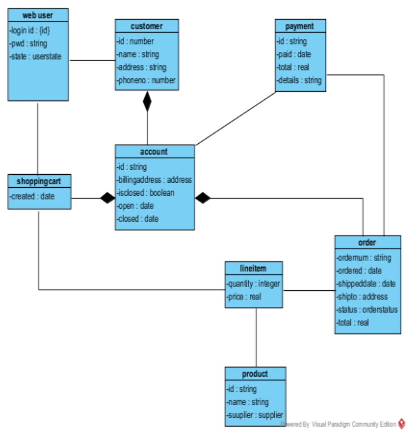
* Event can occur zero to many times
+ Event can occur zero to one time

d) Identify & analyze domain classes



e) Represent use cases and a domain class diagram using Rational Rose



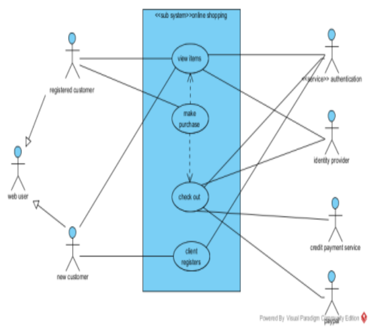

web user
-login id : {id}
-pwd : string
-state : userstate
customer
-id : number
-name : string
-address : string
-phoneno : number
payment
-id : string
-paid : date
-total : real
-details : string
account
-id : string
-billingaddress : address
-isclosed : boolean
-open : date
-closed : date
shoppingcart
-created : date
order
-ordernum : string
-ordered : date
-shippeddate : date
-shipto : address
-status : orderstatus
-total : real
lineitem
-quantity : integer
-price : real
product
-id : string
-name : string
-suuplier : supplier
Powered By Visual Paradigm Community Edition

f) Develop CRUD matrix to represent relationships between use cases and problem domain classes

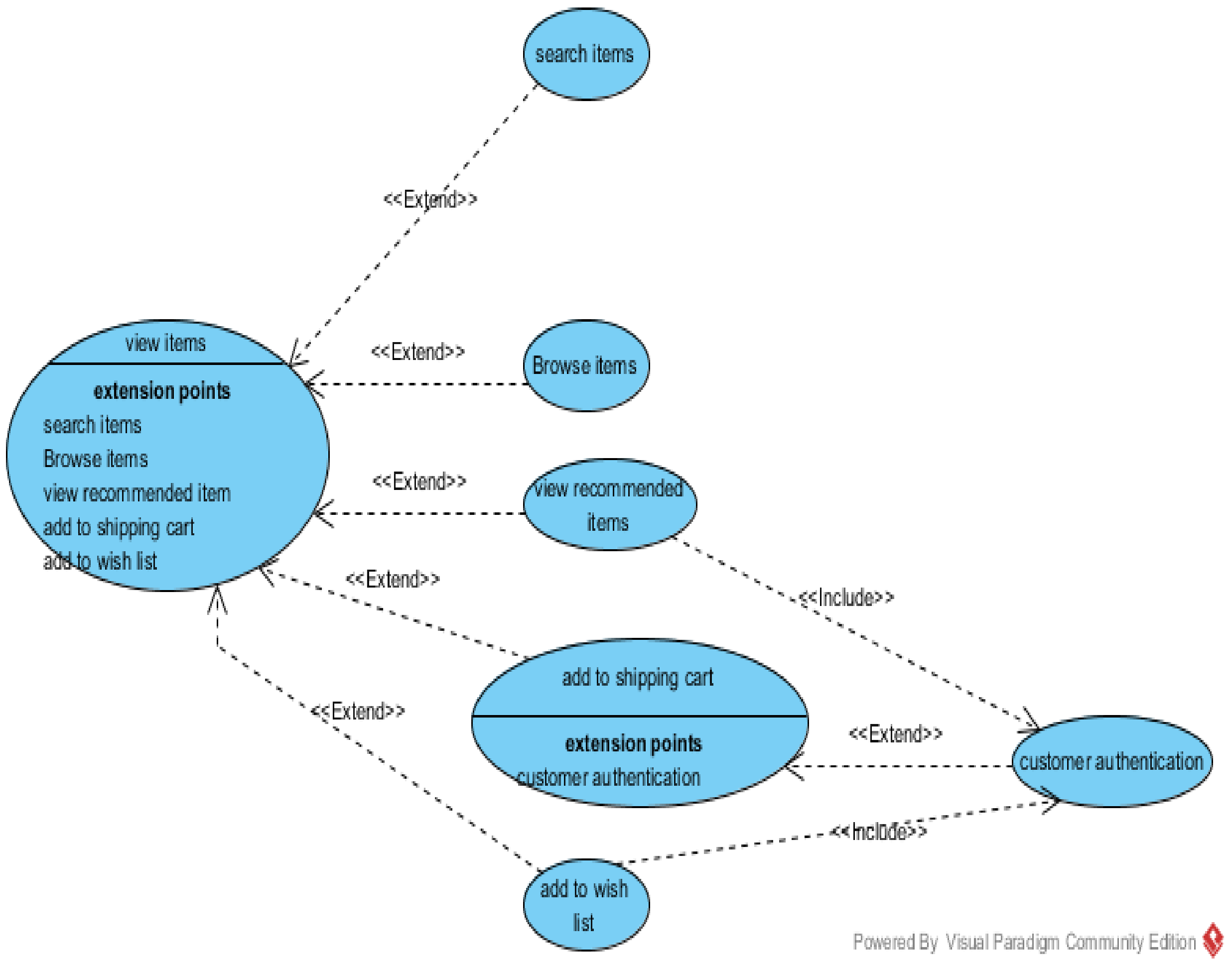
Data Entity	**CRUD**	**Resulting Usecase**
Customer	Create	Add new Customer
	Read/Report	Find Customer
		Generate Customer List
	Update	Update Customer information
	Delete	Delete inactive customer
Order	Create	Create New Order
	Read/Report	View Order
		View Order history
	Update	Update order
	Delete	Cancel order
Inventory Item	Create	Add new inventory item
	Read/Report	View inventory item
		Find inventory item
		Generate inventory item List
	Update	Update inventory item
	Delete	Delete inventory item
Shipment	Create	Create New Shipmnent Request
	Read/Report	View Shipment Details
		Generate Shipment Report
	Update	update shipment request
	Delete	cancel shipment request

Customer supporting System

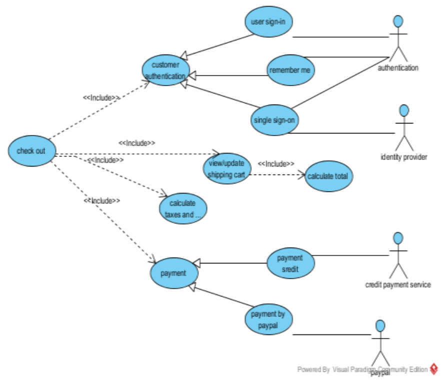
a) **Develop Use case diagrams**



View Item Use Cases

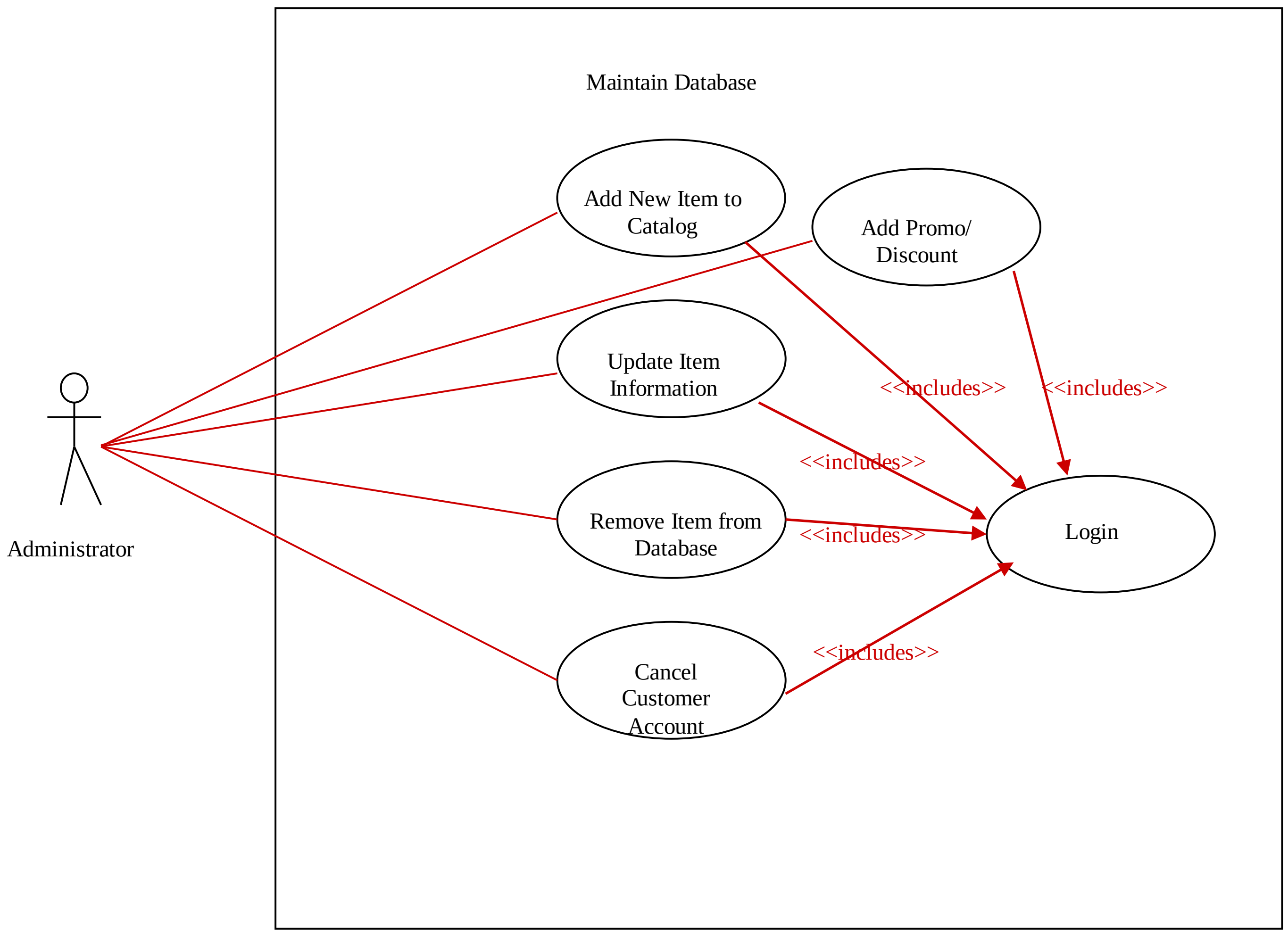


Check out use cases:

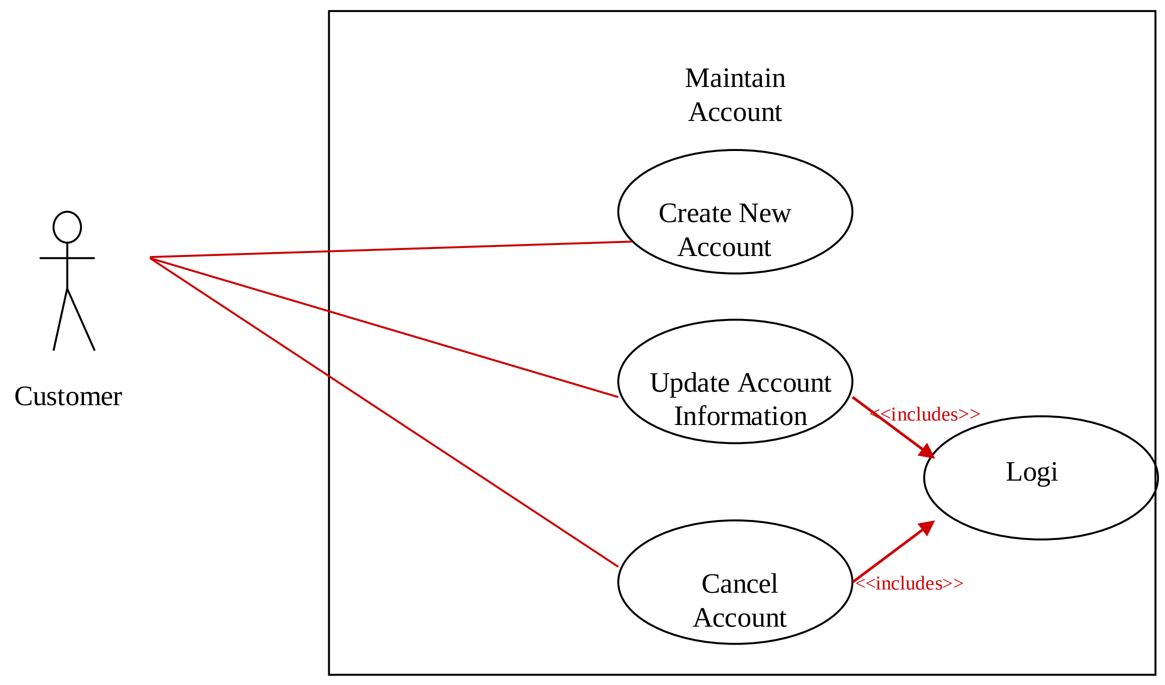


b) Develop elaborate Use case descriptions & scenarios

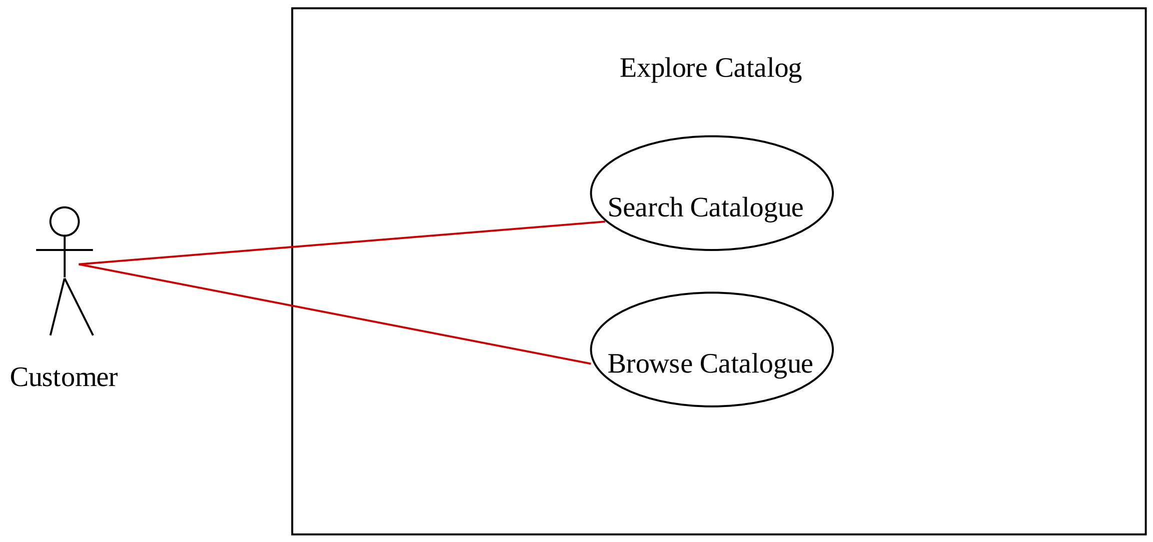
Maintain Shopping Cart



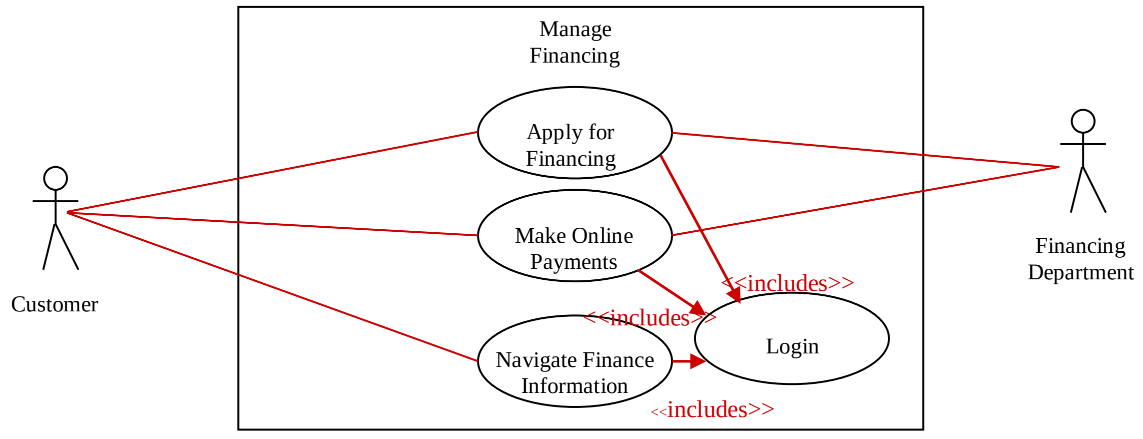
Maintain Account



Explore Catalog



Manage Financing



b) USE CASE DESCRIPTIONS

USE CASE UC1: BROWSE CATALOG

Description: This use case describes how the User can search/browse the e-store catalog.
Primary Actor: User
Stakeholders and Interests:

- **User**: Wants user-friendly interface and fast browsing speed.
  Wants to browse the catalog and add items to the cart successfully.
- **Company**: Wants to satisfy user interests.

Preconditions: None
Success Guarantee (Post Conditions):

- 'Product Screen' displays items and corresponding list prices for a chosen product.
- Item Screen' displays detailed information about an individual item for sale, including a photo, if one is available.
- 'Cart Screen' displays the various items added to the cart, the quantity and list price of each item and the Subtotal.

Basic flow:

1. User opens a web browser, gives the URL for the 'Marvel e-store' website in and clicks on 'Go' button.
2. System launches the web site.
3. User clicks on any product link in the 'Product' list given in top-left corner.
4. System displays 'Category Screen' with the products available for the category chosen in Step 3.
5. User clicks on any product link in 'Products for this Category' list.
6. System displays 'Product Screen' with list of all of the items for the product chosen in Step 5 along with the price of each item and a link labeled 'Add to Cart' in right column of the list.
7. User clicks on any item link in 'Items for this Product' list.
8. System displays 'Item Screen' for the item chosen in Step 7, including a photo if one is available and an 'Add to Cart' link.

Extensions (Alternate Flow):

3a. User navigates to category page of a particular type of product by clicking on any product in the image map located in the center of the page.
4a. User views the next few items from the list of all products in category by clicking on 'Next' link in the bottom right corner of product list and then navigates to 'Product Screen' of a particular product by clicking on that product link in 'Products for this Category' list.

USE CASE UC2: SEARCH CATALOG

Description: This use case describes how the User can search the e-store catalog.
Primary Actor: User
Stakeholders and Interests:

- **User**: Wants user-friendly interface and fast searching speed.
  Wants to find some specific product in catalog and add items to the cart successfully.
- **Company**: Wants to satisfy user interests.

Preconditions: None
Success Guarantee (Post Conditions):

- 'Product Screen' displays items and corresponding list prices for the searched product.
- Item Screen' displays detailed information about an individual item for sale, including a photo, if one is available.
- 'Cart Screen' displays the various items added to the cart, the quantity and list price of each item and the Subtotal.

Basic flow:

1. User opens a web browser, gives the URL for the 'Marvel e-store' website in

   and clicks on 'Go' button.
2. System launches the web site.
3. User enters text in text box next to Search button and clicks on Search button.
4. System displays the matching text products.
5. User clicks on the desired link. User can navigate back to the earlier pages if wishes.

Extensions (Alternate Flow):

3a If search returns no results, System displays the message "No matches found for the entered text".

USE CASE UC3: Create New Account

Description: This use case describes how a new User can register with e-store
P**rimary Actor**: User
Stakeholders and Interests:

- **User**: Wants user-friendly interface and fast searching speed.
  Wants to register and create the account with ease and within a short time.
  Company: Wants to satisfy user interests and validate user information.

Preconditions: E-store website main page is loaded.
Success Guarantee (Post Conditions):

- 'Account verification Screen' lets the user review his/her account details and then successfully register as a user of this site.

Basic flow:

1. The new use click on new 'create new Account link'.
2. The user is at Account Information screen.
3. The new use enters the following details in the Account Information Screen.
   Contact Information:
   a. First Name
   b. Last Name
   c. Street Address
   d. City
   e. State of Province
   f. Country
   g. Postal Code
   h. Telephone Number
   i. E-Mail Credit Card Information
   j. Card Number
   k. Card Type
   l. Card Expiry Date
4. The user clicks on Update and the system validates all the user information and displays the signing information page.
5. User enters the new Username and Password.
6. System validates that the Username is already in use. If not, system displays the new account confirmation page.
7. System sends an e-mail notification about new account creation to User.

USE CASE UC4: Update Account Information

Description: This use case describes how a User can update his account information with e-store
Primary Actor: User
Stakeholders and Interests:

- **User**: Wants user-friendly interface and fast searching speed.
  Wants to update the account with ease and within a short time.
- **Company**: Wants to satisfy user interests and validate user information.

Preconditions: E-store website main page is loaded.
Success Guarantee (Post Conditions):

- 'Account Information Screen' lets the user review his/her account details and then successfully modifying its contents.

Basic flow:

1. User Clicks on the Signin Link
2. System displays the sign In screen
3. User enters the Username and Password
4. System displays the Account Information Page.
5. User can click on the "update Account" link
6. User is at Modifiable Account Information Page.
7. User modifies the account information and exits the page by clicking on “Finish”.
8. System displays the confirmation message “Account Information is updated”.

Miscellaneous:

Steps 1 and 2 in the Basic Flow are part of the ‘Login’ use case. The basic flow of this use case ‘uses’ or ‘includes’ the ‘Login’ use case.

USE CASE UC5: Cancel Account Information

Description: This use case describes how a User can cancel his account information with e-store

P**rimary Actor**: User

Stakeholders and Interests:

- **User**: Wants user-friendly interface and fast searching speed.
  Wants to cancel the account with ease and within a short time.
- **Company**: Wants to satisfy user interests and validate user information.

Preconditions: E-store website main page is loaded.

Success Guarantee (Post Conditions):

- ‘Account Information Screen’ lets the user cancel his account.

Basic flow:

1. User Clicks on the Signing Link.
2. System displays the sign In screen.
3. User enters the Username and Password.
4. System displays the Account Information Page.
5. User can click on the "Cancel Account" link.
6. System displays the confirmation message “Account deleted”.
7. System sends an e-mail to user confirming the cancellation of the account.

USE CASE UC6: Maintain Shopping Cart

Description: This use case describes how an actor can modify items in the shopping cart.
Primary Actor: User
Stakeholders and Interests:

> **User:** Wants to browse/purchase electronic items from the Store.
> **Marvel electronics Owner:** Every user who visits the site or makes a purchase has a direct bearing on the revenue and hence the profitability of the store owner.
> **Pre-Condition:** The actor is on the Cart Screen and have already logged in.
> **Post-Condition:** The user successfully modifies existing items in the cart or adds new items to the cart.

Basic Flow

1. The user clicks on one of the category in the left frame of the screen and navigates to the item he wishes to add to the cart and clicks on the "Add to Cart" link.
2. The system displays the Cart Screen with the all the old items and the newly added item. The subtotal field displays the total cost of the shopping cart.
3. The user repeats steps 3 and 4 for all the items he wants to add to the cart.
4. The user modifies the item quantity for one or multiple items and clicks "Update Cart".
5. The system updates the new quantity and displays the modified line item totals and sub-total to the user.
6. The user clicks the "Remove" link to remove any of the items in the cart.
7. The system deletes the item from the cart and adjusts the sub-total accordingly.

Extensions (or Alternative Flows):

a. User proceeds to adding Items to cart and modifying cart without logging in.
b. If the user enters a non-positive or non-integer quantity the system displays an appropriate error message.
c. If user closes the window without proceeding for payment, the cart is stored in the system for a pre-decided number of days, before getting flushed, so that the user can return to the cart in the future.
d. 'Refresh cart' feature is available for resetting the cart.

Special Requirements:

1. Multiple users should be able to add items to cart simultaneously.

Technology and Data Variation List: None
Frequency of Occurrence: There is a possibility that multiple users will add an item to the same cart simultaneously from different locations.

USE CASE UC7: Maintain Database

Description: This use case describes how the administrator of the system can add and delete items from the catalog and also manage the system users.

Stakeholders and Interests:

Administrator: Wants to add/modify items in the product catalog.

User: Wants updated product catalog.

Owner: Every user who visits the site or makes a purchase has a direct bearing on the revenue and hence the profitability of the store owner.

Pre-Condition: The Marvel electronic store product web page is loaded. The administrator is logged into the system.

Post-Condition: The user successfully manages the users and the catalog.

Basic Flow

1. The system prompts the user to select one of the following two options:
   - Manage Catalog
   - Manage Users
2. If the user selects the "**Manage Catalog**" option, the system prompts the user to select one of the following two options:
   - Add new item
   - Modify existing item, i.e., update or remove item.

   2A. If the user selects the "***Add new item***" option,

   2a. The system prompts the user to select an appropriate category and product (or create a new category/product if one does not exist) to place the item)

   2b. The user select the appropriate category and product.

   2c. The system prompts the user to enter the item details like Item Name, Quantity Available, Price and Item Image.

   2d. The user keys in the requested item details and clicks "Submit".

   2e. The system updates the item in the selected category/product in the database.

   2B. If the user selects the "***Modify Existing Item***" option,

   2a. The system prompts the user to navigate to the appropriate item.

   2b. The user navigates to the item that he wants to modify.

   2c. The user either removes the item from the catalog by clicking "Remove Item" or modifies the Item Name, List Price, Quantity or Item Image and clicks Update.

   2d. The system updates the information in the database.
3. If the user selects the "**Manage Users**" option, the system prompts the user to select one of the following two options:
   - Add User
   - Modify User

   3A. If the user selects the "**Add User**" option,

   3a. The system displays the "Add new user" page to the user.

   3b. The user enters the user details like name, address, etc and selects the access right (normal user/ administrator) of the user and clicks Submit.

   3c. The system updates the new user details in the database.

   3B. If the user selects the "**Modify User**" option,

   3a. The system prompts the admin to search for the user.

   3b. The user searches for the user he wants to modify.

   3c. The system displays the user details to the admin.

   3d. The admin modifies any of the user details like name, address, card details, access rights and clicks Update.

3e. The system updates the details in the database.

Extensions (Alternative Flows):

2A. 2d 1 Incomplete Item Information

If the user fails to enter any of the mandatory item information like Item Name, Quantity and Price then the system displays an appropriate error message to the user.

3A. 3b.1 Incomplete User Information

If the user fails to enter any of the mandatory user information like user Name or Password then the system displays an appropriate error message to the user.

3 Incomplete Selection

If the user does not select any of the options add user or modify user then the system displays an appropriate error message to the user.

USE CASE UC8: Apply For Financing

Description: This use case describes how the User of the system can apply for finance for buying the products.

Primary Actor: User

Stakeholders and Interests:

User: Wants better payment options.
Wants to buy the products on loan basis .
Company: Wants to satisfy user interests.

Preconditions: 'Marvel E-Store' page should be loaded.

Success Guarantee (Post Conditions):

User placed an application successfully.
Application Id is generated by the system.
Confirmation e-mail is sent by the system to the User.

Basic flow:

1. User clicks on "Financing link".
2. System displays the "Financing" main page.
3. User clicks on 'Apply Now' link in the Screen.
4. System displays the 'Application Screen'.
5. User enters the following information.

Applicant Information:

First Name
Last Name
Street Address
City
State of Province
Country
Postal Code
Telephone Number
E-Mail
Current Annual income of household
Any loan rejected before
Installment option preferable(6months,1year,2year)

6. User clicks on 'Submit' button.

7. System displays ‘Confirmation Application Submitted Screen’ to the User with the Application ID.
8. System sends a confirmation e-mail to the User.

USE CASE UC9: Make Online Payments

Description: This use case describes how the User of the system can make payments for the loan.

Primary Actors: User

Stakeholders and Interests:

User: Wants better payment options.
Wants to pay for the loan online.
Company: Wants to satisfy user interests.

Preconditions: ‘Marvel E-Store’ page should be loaded.

Success Guarantee (Post Conditions):

- User is able to pay online successfully.
- A Confirmation Id is generated by the system.
- Confirmation e-mail is sent by the system to the User.

Basic flow:

1. User clicks on “Financing link”.
2. System displays the “Financing” main page.
3. User clicks on ‘Make Payments’ link in the Screen.
4. System displays the ‘Payment Screen’.
5. User enters the following information.
   Applicant Information:
   - First Name
   - Last Name
   - Mode of Payment

   If mode of payment debit/credit card
   - Credit card number
   - Credit expiry Date
   - Card Type

   If mode of payment is cheque
   - Cheque Number
   - Routing Number
   - Bank Name
   - Account Number
6. User clicks on ‘Submit’ button.
7. System displays ‘Confirm the payment information again’ to the User.
8. User click on 'Confirm'.
9. System sends a confirmation e-mail to the User.

Special Requirements

1. If payment is done through the credit/debit card, there is a requirement of consulting the credit/debit company for confirming the account and payment.

USE CASE UC10: Navigating Finance Information

Description: This use case describes how the User of the system can navigate through the financing information.

Primary Actor: User

Stakeholders and Interests:

User: Wants easy accessibility of the financing information.

Company: Wants to satisfy user interests.

Preconditions: 'Marvel E-Store' page should be loaded.

Success Guarantee (Post Conditions):

User is able to browse through the financing information successfully.

User can check his balance.

User can check the history information of his payments.

Basic flow:

1. User clicks on "Financing link".
2. System displays the "Financing" main page.
3. User clicks on 'History' link in the Screen.
4. System displays the 'Payment History Screen'showing the dates and amount of intallments.

Extensions (Alternate Flow):

3a 1. User clicks on 'Balance' link in the Screen.
   2. System displays the 'Balance Payment' screen showing the amount due on the user.

3b 1. User clicks on 'Financing Details' link in the Screen.
   2. System displays the 'Details on loan' screen showing the loan details.

USE CASE UC11: Login/Registration

Description

This use case describes how users gain access to the e-Store system through the login/registration (account creation) process.

Primary Actors

Users (Customers, Administrators)

Stakeholders and Interests

1. User: wants to gain access to the system for any number of reasons (e.g., maintain personal account, check order status, purchase items, administer system, etc.).
2. Marvel Electronics Owner: wants to ensure security of system.
3. Pre-Condition: the user is on the "Sign In" screen.
4. Post-Condition: the user is either logged in or failed to log in and is appropriately notified.

Basic Flow (Returning User, Valid Username/Password)

1. The user browses to the "Sign In" page.
2. The user enters his/her username and password in the returning user section of the "Sign In" screen.
3. The system validates the username and password (successfully) and displays the user's account information page.

Extensions (Alternative Flows):

1. The user browses to the "Sign In" page.
2. The user enters his/her username and password.
3. The system determines that the username or password is invalid and informs the user to try again.

Returning User, Forgotten Username or Password

1. The user has forgotten his/her username, password, or both, and clicks the "Forgot Username/Password?" link
2. The system resets the users account and sends an e-mail notification with the new information
3. The user utilizes the new username/password information to log in following the basic flow

New User

1. The user browses to the "Sign In" page.
2. The user chooses the "New User" link on the "Sign In" page.
3. The user enters his/her account information and chooses a username and password
4. The system validates the information entered
5. The system sends the user an e-mail invitation
6. The user must confirm their new account by clicking the link in the e-mail
7. The user is logged in and his/her account information page is displayed.

System Administrator

System administrators follow the basic flow for this use case when logging in to the system.

Special Requirements

- After three consecutive unsuccessful login attempts, the user's account will be locked and must be reset by a system administrator.

- Users may not login from multiple different computers simultaneously. If this condition is detected, the user will be notified with appropriate warning/error messages.

Technology and Data Variation List

None

Frequency of Occurrence

Users must log in to access their account information, to process a return request, and, optionally, to place an order. The system administrator must log in to administer the system.

USE CASE UC12: Process Returns

Description

This use case describes how customers can request Return Merchandise Authorizations (RMAs) using the e-Store system.

Primary Actors

Customers, Support Personnel, Shipping/Receiving Department, Card Payment Vendor

Stakeholders and Interests

Customer: wants to return an item (or items) purchased through the e-Store system.
Pre-Condition: the customer has purchased an item and wishes to return it.Post-Condition: the customer has received either approval to return the item(s), or a denial of the return request.

Basic Flow (RMA Granted)

1. The customer logs in to the e-Store system.
2. The customer browses to the "Return Products" page and requests an RMA.
3. The customer logs out of the e-Store system (optionally).
4. The e-Store system notifies support personnel of the pending RMA.
5. Support personnel approve the RMA request.
6. The e-Store system notifies the customer of the request approval and sends further instructions via e-mail.
7. The customer ships the item(s) back to Marvel Electronics according to the instructions
8. The receiving department at Marvel Electronics acknowledges the receipt in the e-Store system
9. The e-Store system refunds the customer's payment
10. The e-Store notifies the customer of the receipt and refund via e-mail.

Extensions (Alternative Flows):

RMA Denied

1. The customer logs in to the e-Store system.
2. The customer browses to the "Return Products" page and requests an RMA.
3. The customer logs out of the e-Store system (optionally).
4. The e-Store system notifies support personnel of the pending RMA.
5. Support personnel deny the RMA request.
6. The e-Store system notifies the customer of the request denial via e-mail.

RMA Approved, With Card Payment

1. The customer logs in to the e-Store system.
2. The customer browses to the "Return Products" page and requests an RMA.
3. The customer logs out of the e-Store system (optionally).
4. The e-Store system notifies support personnel of the pending RMA.
5. Support personnel approve the RMA request.

6. The e-Store system notifies the customer of the request approval and sends further instructions via e-mail.
7. The customer ships the item(s) back to Marvel Electronics according to the instructions
8. The receiving department at Marvel Electronics acknowledges the receipt in the e-Store system
9. The e-Store system contacts the card payment vendor to refund the customer's payment to their credit card.
10. The e-Store system notifies the customer of the receipt and refund via e-mail.

Special Requirements
None
Technology and Data Variation List
None
Frequency of Occurrence
N/A

USE CASE UC13: Process Help Request

Description
This use case describes how customers can get service and support using the e-Store system.
Primary Actors
Customers, Support Personnel
Stakeholders and Interests

1. Customer: wants to get help/support with a product, service, or e-Store issue.
2. Marvel Electronics Owner: wants to ensure customer satisfaction.
3. Pre-Condition: none.
4. Post-Condition: The customer has received the desired help/support.

Special Requirements
None
Technology and Data Variation List
None
Frequency of Occurrence
N/A

USE CASE UC14: Accept Payment Online
Description
This use case describes how Financing Department can accept the payment from the e-Store system.
Primary Actors Financing Department
Stakeholders and Interests
Financing Department Easy browsing for payments details.
Basic Flow:

1. Financing Department Personnel can click on Financing Department page.
2. Click on 'E-Payment link'.
3. Click on 'Customer Payment' link.
4. System displays the Customer Payment Page.
5. Financing Department Person can view the details and accept the payment by clicking 'Payment Received'.

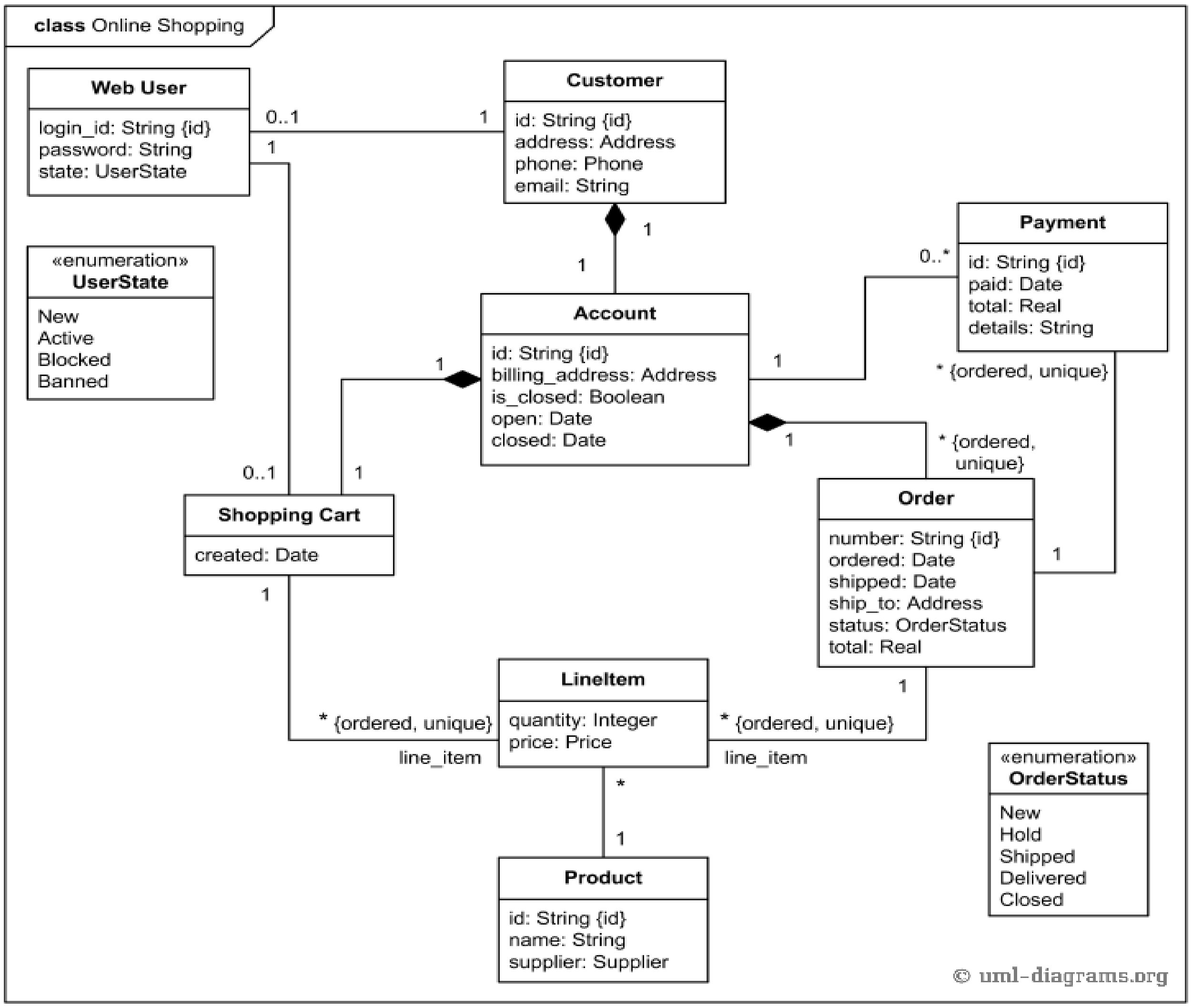
Special Requirements
Financing Department has access to all the credit/debit card companies and banks to confirm the e-payment given by customers.
Technology and Data Variation List
None
Frequency of Occurrence
N/A

c) **Develop prototypes (without functionality):**

A **prototype** is an early sample, model, or release of a product built to test a concept or process or to act as a thing to be replicated or learned from.

- ➢ A prototype is designed to test and try a new design to enhance precision by system analysts and users. Prototyping serves to provide specifications for a real, working system rather than a theoretical one

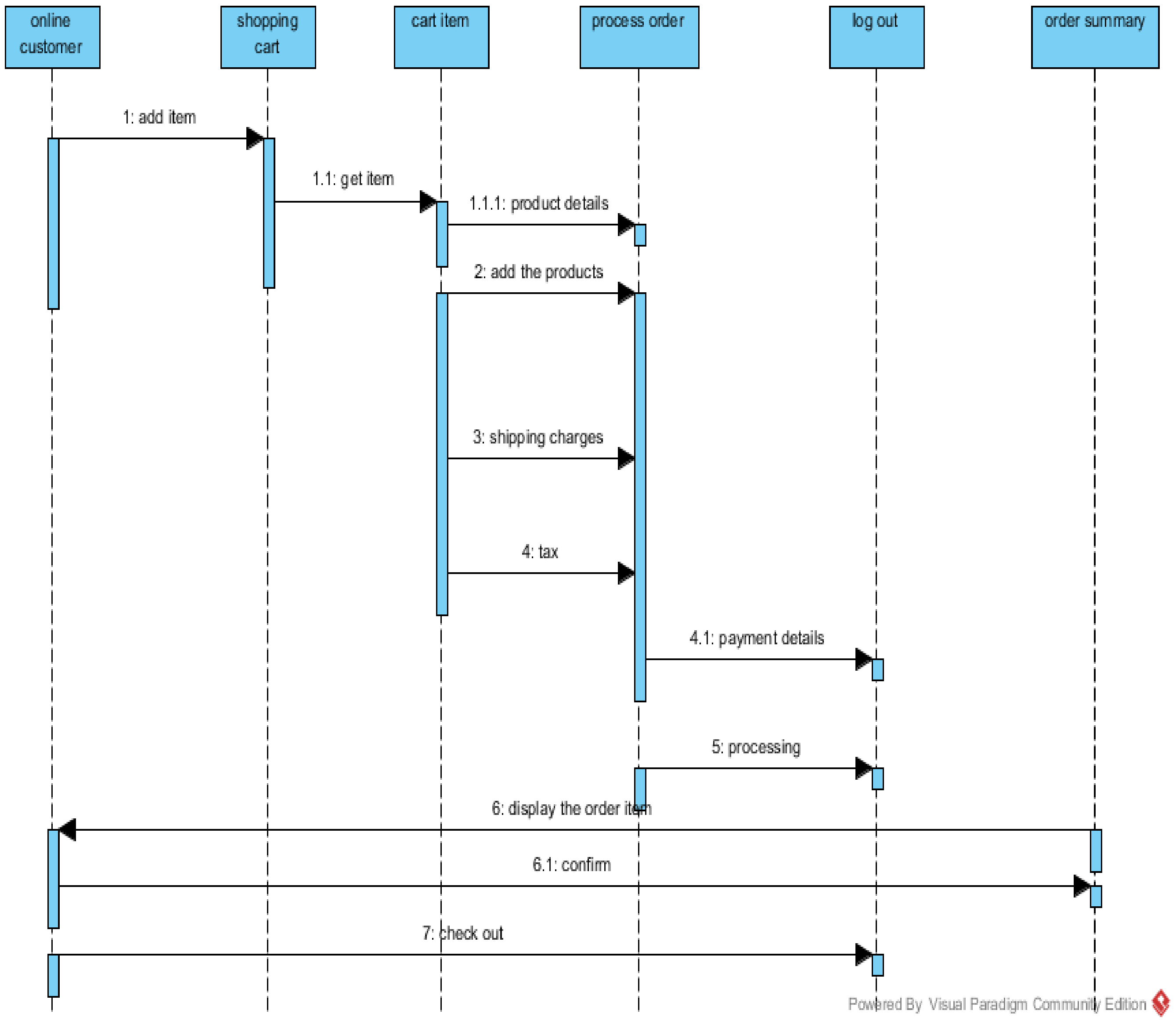
- In some workflow models, creating a prototype (a process sometimes called **materialization**) is the step between the formalization and the evaluation of an idea.
- It is a term used in a variety of contexts, including semantics, design, electronics, and software programming..



d) Develop system sequence diagrams

A **Sequence diagram** is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a Message Sequence Chart.

- A sequence diagram shows object interactions arranged in time sequence.
- Sequence diagram and Collaboration diagrams are Isomorphic diagrams.


online customer
shopping cart
cart item
process order
log out
order summary
1: add item
1.1: get item
1.1.1: product details
2: add the products
3: shipping charges
4: tax
4.1: payment details
5: processing
6: display the order item
6.1: confirm
7: check out
Powered By Visual Paradigm Community Edition

UML VIVA QUESTIONS

1. What is UML?

   Answer:

   UML stands for the Unified Modeling Language.
   It is a graphical language for 1) visualizing, 2) constructing, and 3) documenting *the artifacts* of a system.
   It allows you to create a blue print of all the aspects of the system, before actually physically implementing the system.

2. What are the advantages of creating a model?

   Answer:

   Modeling is a proven and *well-accepted engineering technique* which helps build a model.
   Model is a simplification of reality; it is a blueprint of the actual system that needs to be built.
   Model helps to visualize the system.
   Model helps to specify the structural and behavior of the system.
   Model helps make templates for constructing the system.
   Model helps document the system.

3. What are the different views that are considered when building an object-oriented software system?

   Answer**:** Normally there are 5 views.

   1. Use Case view - This view exposes the requirements of a system.
   2. Design View - Capturing the vocabulary.
   3. Process View - modeling the distribution of the systems processes and threads.
   4. Implementation view - addressing the physical implementation of the system.
   5. Deployment view - focus on the modeling the components required for deploying the system.

4. What are the major three types of modeling used?

   Answer:

   The 3 Major types of modeling are

   1. architectural,
   2. behavioral, and
   3. structural.

5. Name 9 modeling diagrams that are frequently used?

   Answer:

   9 Modeling diagrams that are commonly used are

   1. Use case diagram
   2. Class Diagram
   3. Object Diagram
   4. Sequence Diagram
   5. statechart Diagram
   6. Collaboration Diagram
   7. Activity Diagram
   8. Component diagram
   9. Deployment Diagram.

6. How would you define Architecture?

   Answer:

   Architecture is not only taking care of the structural and behavioral aspect of a software system but also taking into account the software usage, functionality, performance, reuse, economic and technology constraints.

7. What is SDLC (Software Development Life Cycle)?

   Answer**:**

   SDLC is a system including processes that are

1. Use case driven,
2. Architecture centric,
3. Iterative, and
4. Incremental.

8 What are Relationships?

Answer**:**

There are different kinds of relationships:

1. Dependencies,
2. Generalization, and
3. Association.

Dependencies are relationships between two entities.
A change in specification of one thing may affect another thing.
Most commonly it is used to show that one class <u>uses</u> another class as an argument in the signature of the operation.

Generalization is relationships specified in the class subclass scenario, it is shown when one entity inherits from other.
Associations are *structural relationships* that are:

4. a room has walls,
5. Person works for a company.

Aggregation is a type of association where there is a has a relationship.
As in the following examples: A room has walls, or if there are two classes room and walls then the relationship is called a association and further defined as an aggregation.

9 . How are the diagrams divided?

Answer**:**

The nine diagrams <u>are divided</u> into static diagrams and dynamic diagrams.

10. Static Diagrams (Also called Structural Diagram):

Answer:

The following diagrams are static diagrams.

6. Class diagram,
7. Object diagram,
8. Component Diagram,
9. Deployment diagram.

11. Dynamic Diagrams (Also called Behavioral Diagrams):

Answer:

The following diagrams are dynamic diagrams.

10. Use Case Diagram,
11. Sequence Diagram,
12. Collaboration Diagram,
13. Activity diagram,
14. Statechart diagram.

12 What are Messages?

Answer:

A message is the specification of a communication, when a message <u>is passed</u> that results in action that is in turn an executable statement.

13 What is an Use Case?

Answer:

A use case specifies the behavior of a system or a part of a system.

Use cases are used to capture the behavior that need to be developed.

It involves the interaction of actors and the system.

14 What is modeling? What are the advantages of creating a model?

Answer

Model is a simplification of reality; it is a blueprint of the actual system that needs to be built. Model helps to visualize the system. Model helps to specify the structural and behavior of the system. Model helps make templates for constructing the system.

15 What are the different views that are considered when building an object-oriented software system?

Answer

Normally there are 5 views.
Use Case view - This view exposes the requirements of a system.

- Design View - Capturing the vocabulary.
- Process View - modeling the distribution of the systems processes and threads.
- Implementation view - addressing the physical implementation of the system.

Deployment view - focus on the modeling the components required for deploying the system.

16. What is modeling? What are the advantages of creating a model?

Answer

Model is a simplification of reality; it is a blueprint of the actual system that needs to be built. Model helps to visualize the system. Model helps to specify the structural and behavior of the system. Model helps make templates for constructing the system.

17 What are the different views that are considered when building an object-oriented software system?

Answer

Normally there are 5 views.
Use Case view - This view exposes the requirements of a system.

- Design View - Capturing the vocabulary.
- Process View - modeling the distribution of the systems processes and threads.
- Implementation view - addressing the physical implementation of the system.

Deployment view - focus on the modeling the components required for deploying the system.

18 What are diagrams?

Answer

Diagrams are graphical representation of a set of elements most often shown made of things and associations.

19 What are the major three types of modeling used?

Answer

Major three types of modeling are structural, behavioral, and architectural.

20. What is Architecture?

Answer

Architecture is not only taking care of the structural and behavioral aspect of a software system but also taking into account the software usage, functionality, performance, reuse, economic and technology constraints.

21 What are Messages?

Answer

A message is the specification of a communication, when a message is passed that results in action that is in turn an executable statement.

22 Can you explain 'Extend' and 'Include' in use cases?

Answer

- Include: Include relationship represents an invocation of one use case by the other. If you think from the coding perspective its like one function been called by the other function.
- Extend: This relationship signifies that the extending use case will work exactly like the base use case only that some new steps will inserted in the extended use case.

23 How do we represent private, public and protected in class diagrams?

Answer

In order to represent visibility for properties and methods in class diagram we need to place symbols next to each property and method as shown in figure 'Private, Public and Protected'. '+' indicates that it's public properties/methods. '-'indicates private properties which means it can not be accessed outside the class. '#' indicate protected/friend properties. Protected properties can only be seen within the component and not outside the component.

24 Can you explain sequence diagrams?

Answer

Sequence diagram shows interaction between objects over a specific period time. The message flow is shown vertically in waterfall manner i.e. it starts from the top and flows to the bottom. Dashed lines represent the duration for which the object will be live. Horizontal rectangles on the dashed lines represent activation of the object. Messages sent from a object is represented by dark arrow and dark arrow head. Return message are represented by dotted arrow.

25 Can you explain primary and secondary actors?

Answer

Actors are further classified in to two types primary and secondary actors. Primary actors are the users who are the active participants and they initiate the user case, while secondary actors are those who only passively participate in the use case.

26 Can you explain use case diagrams?

Answer

Use case diagram answers what system does from the user point of view. Use case answer ‘What will the system do?’. Use cases are mainly used in requirement document to depict clarity regarding a system. There are three important parts in a use case scenario, actor and use case.

27 Explain how sequence diagram differ from component diagram?

Answer

1.Component diagrams are used to illustrate complex systems,they are building blocks so a component can eventually encompass a large portion of a system, but sequence diagrams are not for it

28 What is the difference between an image and a map

Answer

Image:An image is an exact replica of the contents of a storage device stored on a second storage device. Map:A file showing the structure of a program after it has been compiled. The map file lists

29 Whatis the difference between an API and a framework?

Answer

Furthermore, an API is often a finished, working component which is readily used (maybe with some minor additions). Frameworks are in no instance ready out-of-the-box and aim to take care some of the ...

30 Who developed uml?

Answer

No, UML was not developed by Rational Software(was developed by the Object Management Group), although some of the people on the OMG committee do/did work for RS (now owned by IBM, in the Rational division).

30. Define modeling in UML and it advantages.

Answer

- Model is a simplification of reality.
- Blueprint of the actual system.
- Specify the structural and behavior of the system.
- Templates for designing the system.
- Helps document the system.

31. If you want to plan project activities such as developing new functionalities or test cases, which
of the following OOAD artifacts is the most useful?
(a) Sequence diagrams
(b) Use cases
(c) Domain model
(d) Package diagrams Ans : b

32. Which of the following is iterative, incremental, use case driven and architecture centric?
(a) V-method
(b) UML
(c) Component Based Development
(d) RUP(Rational Unified Process) Ans : d

33. What is true about UML stereotypes?

(a) A stereotype is used for extending the UML language.

(b) A stereotyped class must be abstract.

(c) The stereotype {frozen} indicates that the UML element cannot be changed.

(d) UML Profiles can be stereotyped for backward compatibility. Ans : a

34. Consider a beverage machine. If the actor is 'customer', and the scope is 'machine', what is most likely to be found in the main scenario of the use case 'get drink'?

(a) - enter choice
- if drink available then show price
- put in coins
- if paid enough then deliver drink

(b) - customer enters choice
- machine shows price
- customer puts in coins
- machine delivers drink

(c) - enter choice
- show price
- put in coins
- deliver drink

(d) - machine sends price to LCD display
- customer put coins in slot
- coin mechanism verifies amount and tells machine controller
- machine controller activates boil Ans : b

35. What can UML interfaces be used for?

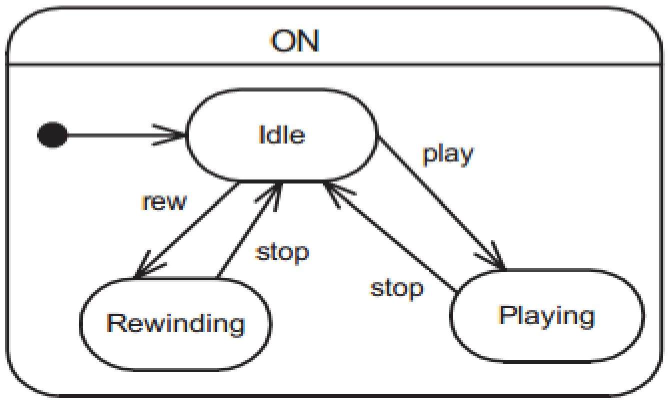
(a) to provide concrete classes with the stereotype <<interface>>

(b) to program in Java and C++, but not in C#

(c) to define executable logic that can be reused in several classes

(d) to specify required services for types of objects Ans : d

36. What is correct about the following State Diagram?



(a) ‘ON’ is a concurrent state.

(b) This State Diagram is invalid because it contains no final state.

(c) ‘play’, ‘stop’ and ‘rew’ are actions.

(d) ‘ON’ is a superstate. Ans : d

37. If you need to show the physical relationship between software components and the hardware

in the delivered system, which diagram can you use?

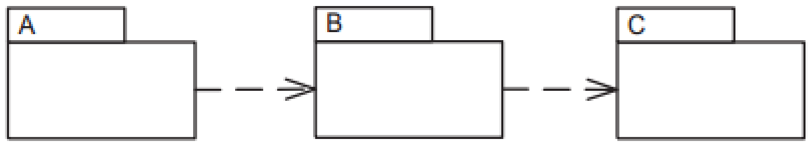
(a) component diagram

(b) deployment diagram

(c) class diagram

(d) network diagram Ans : b

38. What is a true statement about the following packages?



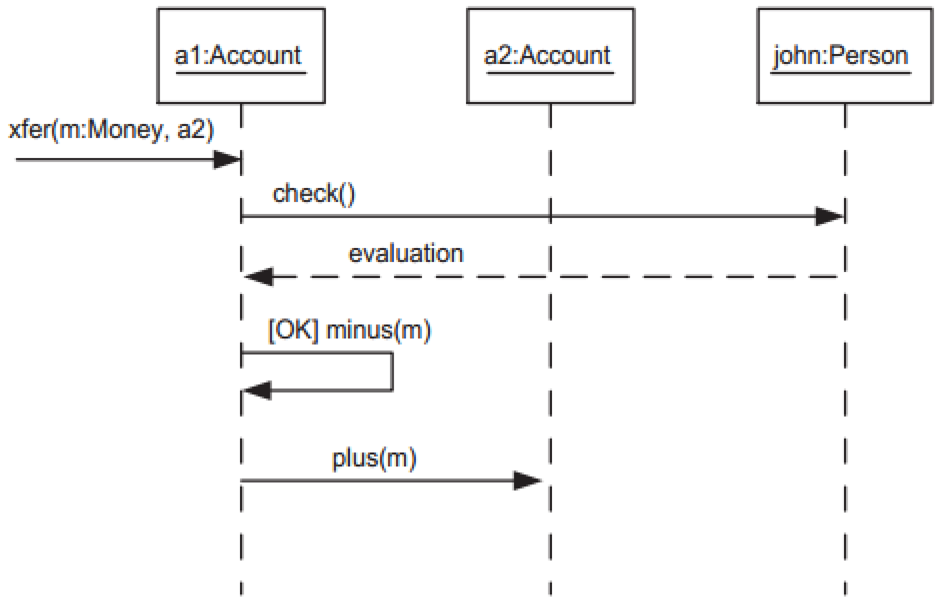
(a) If package C changes, package B must be inspected for necessary changes, and if there are any, package A may have to be adapted as well.

(b) If package B changes, package A and package C must be inspected for necessary changes.

(c) Packages should be designed so that a change in one package does not have an effect to other packages.

(d) Packages should be designed so that a change in one package does not have an effect to other packages. Ans : a

39. Given the following diagram, which method(s) should be implemented for the Account class?



(a) xfer()

(b) xfer(), plus(), minus()

(c) check(), plus(), minus()

(d) xfer(), evaluation(), plus(), minus() Ans : b

40. Which diagram is NOT commonly used for illustrating use cases?

(a) system sequence diagram

(b) activity diagram

(c) use case diagram

(d) collaboration diagram Ans : d

41. Inheritance in object-oriented modelling can be used to

(a). generalize classes

(b). specialize classes

(c). generalize and specialize classes

(d). create new classes Ans : c

42. An object is considered an external entity in object-oriented modelling

(a). its attributes are invariant during operation of the system

(b). its attributes change during operation of the system (c). it has numerous attributes

(d). it has no attributes relevant to the system   Ans : a

43. Given a word statement of problem potential operations appropriate for objects are identified by selecting

(a). verb phrases in the statement

(b). noun phrases in the statement

(c). adjectives in the statement

(d). adverbs in the statement   Ans : a

44. In object-oriented design

(a). operations and methods are identical

(b). methods specify algorithms whereas operations only state what is to be done

(c). methods do not change values of attributes

(d). methods and constructor are same   Ans : d