Ritabroto Ganguly 001910501090 BCSE-II

OOP Assignment 5

1. There are number of students. For every student roll (unique), name is to be stored. For each subject, subject code and name is to be stored. A student can opt for number of subjects. System should be able to maintain student list, subject list and will be able to answer: i) which student has selected which subjects and ii) for a subjects who are the students.
   Design the classes and implement. For list consider memory data structure.

```
#include <iostream>

#include <string>

#include <vector>

using namespace std;


class student;

class Key;


class subject{

  static int codes;

  int code;

  string name;

  vector<student> students;


  public:
```

```
  subject(string name):code(codes),name(name){codes++;}


  int get_code(){return code;}


  string get_name(){return name;}


/*Extra key parameter used following the Passkey idiom to restrict access
        of this function*/
  void add_student(const Key&,student *stu){
    students.push_back(*stu);
  }


  friend ostream& operator<<(ostream& o,const subject& s){
     o<<s.code<<"\t"<<s.name<<endl;
     return o;
  }


  vector<student> get_students(){return students;}
  void show_students();
};
int subject::codes = 1;
vector<subject> subjects;
```

```
class student{
  static int rolls;
  int roll;
  string name;
  vector<subject> subjects;

  public:
  student(string& name):roll(rolls),name(name){rolls++;}

  int get_roll(){return roll;}

  string get_name(){return name;}

  friend ostream& operator<<(ostream& o,const student& s){
    o<<s.roll<<"\t"<<s.name<<endl;
    return o;
  }

  void add_subject();

  vector<subject> get_subjects(){return this->subjects;}
  void show_subjects(){
```

```
    cout<<"subjects: "<<subjects.size()<<endl;
    for(int i=0;i<this->subjects.size();i++)
       cout<<subjects[i];
  }
};
int student::rolls = 1;
vector<student> students;

void subject::show_students(){
  cout<<"students: "<<subject::students.size()<<endl;
  for(int i=0;i<subject::students.size();i++)
      cout<<subject::students[i];
}

/* Key class for Passkey Idiom */
class Key{
  Key(){} // default ctor private to restrict access to Key
  Key(const Key&){} // copy ctor private to restrict access to Key

  // grant access to add_subject() method of student to create object of
        Key
  friend void student::add_subject();
};
```

```
void student::add_subject(){
   while(true){
    cout<<"Enter subject name for Name:"<<this->name<<", Roll:"<<this-
        >roll<<" : ";
    string x;
    getline(cin,x);
    int subs = ::subjects.size();
    subject *sub = nullptr;
    for(int i=0;i<subs;i++)
       if(::subjects[i].get_name()==x)
         sub = &(::subjects[i]);

    if(sub!=nullptr){
      student::subjects.push_back(*sub);
      sub->add_student(Key(),this);
      return;
    }else{
       cout<<x<<" subject is not available"<<endl;
    }
   }
}
```

```
//string x[] = {"English","Bengali","Hindi"};
//string y[] = {"rg","sg"};

int main()
{
 int subs = 3,stus = 2;
 cout<<"Enter number of subjects: ";
 cin>>subs;
 cout<<"Enter number of students: ";
 cin>>stus;
 cin.ignore();

 for(int i=0;i<subs;i++){
   // subjects.push_back(subject(x[i]));
  string s;
  cout<<"Enter subject name: ";
  getline(cin,s);
  subjects.push_back(subject(s));
 }
 for(int i=0;i<stus;i++){
    //students.push_back(student(y[i],100*(i+1)));
    if(i>0)
```

```
      cin.ignore();
    string s;
    cout<<"Enter student name: ";
    getline(cin,s);
    students.push_back(student(s));
    int add = 0;
    cout<<"Add subject for "<<s<<"?(yes=1, no=0): ";
    cin>>add;
    while(add==1){
      cin.ignore();
      students[i].add_subject();
      cout<<"Add more subject(s) for "<<s<<"?(yes=1, no=0): ";
      cin>>add;
    }
 }

cout<<"showing subject-wise students:"<<endl;
 for(int i=0;i<subs;i++){
    cout<<subjects[i];
    subjects[i].show_students();
    cout<<"------>>"<<endl;
 }
```

```
 cout<<"--------------------------------"<<endl;
cout<<"showing student-wise subjects:"<<endl;
 for(int i=0;i<stus;i++){
    cout<<students[i];
    students[i].show_subjects();
    cout<<"------>>"<<endl;
 }
 return 0;
}
```

```
Enter subject name: beng
Enter subject name: eng
Enter subject name: hin
Enter student name: rg
Add subject for rg?(yes=1, no=0): 1
Enter subject name for Name:rg, Roll:1 : en
en subject is not available
Enter subject name for Name:rg, Roll:1 : eng
Add more subject(s) for rg?(yes=1, no=0): 1
Enter subject name for Name:rg, Roll:1 : beng
Add more subject(s) for rg?(yes=1, no=0): 0
Enter student name: sg
Add subject for sg?(yes=1, no=0): 1
Enter subject name for Name:sg, Roll:2 : hin
Add more subject(s) for sg?(yes=1, no=0): 0
showing subject-wise students:
1       beng
students: 1
1       rg
------>>
2       eng
students: 1
1       rg
------>>
3       hin
students: 1
2       sg
------>>
----------------------------------
showing student-wise subjects:
1       rg
subjects: 2
2       eng
1       beng
------>>
2       sg
subjects: 1
3       hin
```

2. In a library, for each book book-id, serial number (denotes copy number of a book), title, author, publisher and price are stored. Book-id and serial number together will be unique identifier for a book. Members are either student or faculty. Each member has unique member-id. Name, e-mail, address are also to be stored. For any transaction (book issue or return), members are supposed to place transactions slip. User will submit member-id, book-id, and serial number (only for book return). While processing a transaction, check the validity of the member. While issuing, availability of a copy of the book is to be checked. While returning a book, it is to be checked whether this copy was issued to the member or not. A student member can have 2 books issued at a point of time. For faculty members it is 10. Transaction information is to be stored like date of transaction, member-id, book-id, serial number, returned or not. An entry is made when book is issued and updated when the book is returned.

Design the classes and implement. For list consider memory data structure.

```
#include <iostream>

#include <string>

#include <vector>

using namespace std;


class book{

  static int ids;

  static int serials;

  int id;

  int serial;

  string title,author,publisher;
```

```
float price;

int issued;

int memid;//needed to track which member has issued it currently for
      member::return_book()

public:

book(const string& title,const string& author,const string& pub,float
      price):title(title),author(author),publisher(pub),id(ids),serial(serials),
      price(price),issued(0),memid(0){ids++;serials++;}

int get_memid() const {return memid;}

void set_issued(int i,int m=0){issued = i; memid = m;}

int get_issued() const {return issued;}

int get_id() const {return id;}

int get_serial() const {return serial;}


int operator==(const book& b){

  if(id==b.id && serial==b.serial)

    return 1;

  return 0;

}

friend ostream& operator<<(ostream& o,const book& t){

    o<<t.id<<" "<<t.serial<<" "<<t.title<<" "<<t.author<<"
      "<<t.publisher<<" "<<t.price<<" "<<t.issued<<endl;

    return o;

}
```

```
};
int book::ids = 1;
int book::serials = 1;
vector<book*> all_books;
book* find_book(const book& b){
  for(int i=0;i<all_books.size();i++)
    if(*all_books[i]==b)
      return all_books[i];
  return nullptr;
}

class Key;
class transaction{
  string dot;
  int memid;
  int bookid;
  int serial;
  int issued;
  static vector<transaction> all_transactions;
  transaction(const string& dot,int memid,int bookid,int serial,int issued):
  dot(dot),memid(memid),bookid(bookid),serial(serial),issued(issued)
      {all_transactions.push_back(*this);}
  public:
```

```
  static transaction* check_member(const Key&,const string& dot,int
       memid,int bookid,int serial,int issued);

  friend ostream& operator<<(ostream& o,const transaction& t){

     o<<"date: "<<t.dot<<endl;

     o<<"member id: "<<t.memid<<endl;

     o<<"book id: "<<t.bookid<<endl;

     o<<"serial: "<<t.serial<<endl;

     o<<"issued: "<<(t.issued ? "yes" : "no")<<endl;

     o<<"**********************************"<<endl;

     return o;

  }

  static void show_all_transactions() {

    for(int i=0;i<all_transactions.size();i++)

      cout<<all_transactions[i];

  }

};

vector<transaction> transaction::all_transactions;


class member{

 protected:

 virtual void make_class_abstract() = 0;

 static int ids;

 int max_books;
```

```
int id;

int books;

string name;

string email;

vector<transaction> trans_slip;

member(const string& name,const string& email,int
      m):name(name),email(email),books(0),max_books(m){id=ids;ids++;}

public:

void show_slip(){

  for(int i=0;i<trans_slip.size();i++)

     cout<<trans_slip[i];

}

int get_id() const {return id;}

string get_name() const {return name;}

string get_email() const {return email;}

void set_email(const string& s){email = s;}

int get_books() const {return books;}

int get_max_books() const {return max_books;}

void issue_book(book&);

void return_book(book&);

friend ostream& operator<<(ostream& o,const member& t){

    o<<t.id<<" "<<t.name<<" "<<t.email<<" "<<t.books<<endl;

    return o;
```

```
  }
};
int member::ids = 1;
vector<member*> all_members;
transaction* transaction::check_member(const Key&,const string& dot,int
        memid,int bookid,int serial,int issued){
     for(int i=0;i<all_members.size();i++)
        if(all_members[i]->get_id()==memid)
          return new transaction(dot,memid,bookid,serial,issued);
     return nullptr;
}

class student: public member{
  void make_class_abstract(){cout<<"Memeber is abstract, student is
        not"<<endl;}
  public:
  student(const string& name,const string& email):member(name,email,1)
        {}
};

class faculty: public member{
  void make_class_abstract(){cout<<"Memeber is abstract, faculty is
        not"<<endl;}
  public:
```

```
  faculty(const string& name,const string& email):member(name,email,2){}
};


/* Key class following passkey idiom to restrict access to
        transaction::all_transactions vector from
        transaction::check_member()*/
class Key{
   Key(){}
   Key(const Key&){}

   friend void member::issue_book(book&);
   friend void member::return_book(book&);
};
  void member::issue_book(book& b){
  //   cout<<"books="<<books<<" max_books="<<max_books<<"
        issued="<<b.get_issued()<<endl;
    cout<<*this;
    if(books<get_max_books()){
      if(find_book(b)==nullptr){
        cout<<"Invalid book id"<<endl;
        cout<<"------------------------------"<<endl;
        return;
      }
      cout<<b;
```

```
  if(b.get_issued()==1){
    cout<<"Book is unavailable for the moment"<<endl;
    cout<<"------------------------------"<<endl;
    return;
  }
  string s;
  cout<<"Enter date of issue: ";
  cin>>s;
  transaction* t =
     transaction::check_member(Key(),s,id,b.get_id(),b.get_serial(),1);//
     check if member id is valid
  if(t){
    trans_slip.push_back(*t);
    free(t);
    b.set_issued(1,this->id);
    books++;
    cout<<"ISSUED"<<endl;
  }else
    cout<<"Invalid member id"<<endl;
}else{
  cout<<b;
  cout<<"No issues left"<<endl;
}
```

```
  cout<<"-----------------------------"<<endl;
}
void member::return_book(book& b){
  cout<<*this;
  if(books>0){
    if(find_book(b)==nullptr){
      cout<<"Invalid book id"<<endl;
      cout<<"-----------------------------"<<endl;
      return;
    }
    cout<<b;
    if(b.get_issued()==0){
      cout<<"Book has not been issued"<<endl;
      cout<<"-----------------------------"<<endl;
      return;
    }
    if(b.get_memid() != id){
      cout<<"Book is not yours to return"<<endl;
      cout<<"-----------------------------"<<endl;
      return;
    }
    string s;
```

```
    cout<<"Enter date of return: ";
    cin>>s;
    transaction* t =
       transaction::check_member(Key(),s,id,b.get_id(),b.get_serial(),0);//
       check if member id is valid
    if(t){
      trans_slip.push_back(*t);
      free(t);
      b.set_issued(0);
      books--;
      cout<<"RETURNED"<<endl;
    }else
      cout<<"Invalid member id"<<endl;
  }else{
    cout<<b;
    cout<<"Can't return book"<<endl;
  }
  cout<<"------------------------------"<<endl;
 }

int main() {
 cout<<"2 members entered by default"<<endl;
 member* s1 = new student("student","student@gmail.com");
```

```
member* f1 = new faculty("teacher","teacher@gmail.com");
all_members.push_back(s1);
all_members.push_back(f1);
cout<<*s1<<*f1;

cout<<"3 books entered by default"<<endl;
book *b;
for(int i=0;i<3;i++){
  string t = "title" + to_string(i);
  string a = "author" + to_string(i);
  string p = "publisher" + to_string(i);
  b = new book(t,a,p,(i+1)*100);
  all_books.push_back(b);
  cout<<*b;
}
cout<<"---------------------------"<<endl;

s1->issue_book(*all_books[0]);
s1->issue_book(*all_books[1]);
f1->issue_book(*all_books[0]);
f1->issue_book(*all_books[1]);
s1->return_book(*all_books[1]);
```

```
  s1->return_book(*all_books[0]);
  s1->issue_book(*all_books[2]);
  f1->issue_book(*all_books[0]);

  for(int i=0;i<all_books.size();i++)
    cout<<*all_books[i];
  cout<<"--------------------------------"<<endl;
  for(int i=0;i<all_members.size();i++){
    cout<<*all_members[i];
    all_members[i]->show_slip();
    cout<<"------------------------------------------------------"<<endl;
  }
  transaction::show_all_transactions();
  cout<<"------------------------------------------------------"<<endl;

  return 0;
}
```

```
2 members entered by default
1 student student@gmail.com 0
2 teacher teacher@gmail.com 0
3 books entered by default
1 1 title0 author0 publisher0 100 0
2 2 title1 author1 publisher1 200 0
3 3 title2 author2 publisher2 300 0
------------------------------
1 student student@gmail.com 0
1 1 title0 author0 publisher0 100 0
Enter date of issue: 1/1/1
ISSUED
--------------------------------
1 student student@gmail.com 1
2 2 title1 author1 publisher1 200 0
No issues left
--------------------------------
2 teacher teacher@gmail.com 0
1 1 title0 author0 publisher0 100 1
Book is unavailable for the moment
--------------------------------
2 teacher teacher@gmail.com 0
2 2 title1 author1 publisher1 200 0
Enter date of issue: 2/2/2
ISSUED
--------------------------------
1 student student@gmail.com 1
2 2 title1 author1 publisher1 200 1
Book is not yours to return
--------------------------------
1 student student@gmail.com 1
1 1 title0 author0 publisher0 100 1
Enter date of return: 2/2/2
RETURNED
--------------------------------
1 student student@gmail.com 0
3 3 title2 author2 publisher2 300 0
Enter date of issue: 3/3/3
```

```
1 student student@gmail.com 0
3 3 title2 author2 publisher2 300 0
Enter date of issue: 3/3/3
ISSUED
--------------------------------
2 teacher teacher@gmail.com 1
1 1 title0 author0 publisher0 100 0
Enter date of issue: 4/4/4
ISSUED
--------------------------------
1 1 title0 author0 publisher0 100 1
2 2 title1 author1 publisher1 200 1
3 3 title2 author2 publisher2 300 1
--------------------------------
1 student student@gmail.com 1
date: 1/1/1
member id: 1
book id: 1
serial: 1
issued: yes
**********************************
date: 2/2/2
member id: 1
book id: 1
serial: 1
issued: no
**********************************
date: 3/3/3
member id: 1
book id: 3
serial: 3
issued: yes
**********************************
------------------------------------------------------------
2 teacher teacher@gmail.com 2
date: 2/2/2
member id: 2
book id: 2
issued: yes
**********************************
date: 4/4/4
member id: 2
book id: 1
serial: 1
issued: yes
**********************************
------------------------------------------------------------
date: 1/1/1
member id: 1
book id: 1
serial: 1
issued: yes
**********************************
date: 2/2/2
member id: 2
book id: 2
serial: 2
issued: yes
**********************************
date: 2/2/2
member id: 1
book id: 1
serial: 1
issued: no
**********************************
date: 3/3/3
member id: 1
book id: 3
serial: 3
issued: yes
**********************************
date: 4/4/4
member id: 2
book id: 1
serial: 1
issued: yes
```

3. Employee has unique emp-id, name, designation and basic pay. An employee is either a permanent one or contractual. For permanent employee salary is computed as basic pay+ hra (30% of basic pay) + da (80% of basic pay). For contractual employee it is basic pay + allowance (it is different for different contractual employee). An employee pointer may point to either of the two categories and accordingly the salary has to be created.

Design the classes and implement.

```
#include <iostream>

#include <string>

#include <vector>

using namespace std;


class employee{

  protected:

  int id;

  string name;

  string des;

  float basic;

  public:

  employee(string name,string des,float
        basic):name(name),des(des),basic(basic){}

  virtual float salary() = 0;

};
```

```
class perm_emp : public employee{
  float hra,da;
  public:
  perm_emp(string name,string des,float basic):employee(name,des,basic)
        {}
  float salary(){
    hra = basic*0.3;
    da = basic*0.8;
    return (basic + hra + da);
  }
  float get_hra() const {return hra;}
  float get_da() const {return da;}
};

class temp_emp: public employee{
  float allowance;
  public:
  temp_emp(string name,string des,float basic,float
        allowance):allowance(allowance),employee(name,des,basic){}

  float salary() {return (basic + allowance);}
  float get_allowance() const {return allowance;}
```

```
};

int main()
{
 employee *pe = new perm_emp("perm","engg",10000000);
 employee *te = new temp_emp("perm","engg",1000000,150000);

 cout.precision(3);

 cout<<"Permanent employee salary: "<<fixed<<pe->salary()<<endl;
 cout<<"Temporary employee salary: "<<fixed<<te->salary()<<endl;

 return 0;
}
```

```
Permanent employee salary: 21000000.000
Temporary employee salary: 1150000.000
```

4. Each cricketer has name, date of birth and matches played. Cricketer may be a bowler or batsman. For a bowler, number of wickets taken, average economy is stored. For a batsman, total runs scored, average score is stored. A double wicket pair is formed taking a bowler and a batsman. An all-rounder is both a bowler and batsman. Support must be there to show the details of a cricketer, bowler, batsmen, all-rounder and the pair.

Design the classes and implement.

```
#include <iostream>

#include <vector>

#include <string>

using namespace std;


class cricketer{

  protected:

  string name;

  string dob;

  int matches;

  void print_basic_details() const {

    cout<<"Name: "<<name<<endl;;

    cout<<"DOB: "<<dob<<endl;;

    cout<<"Total matches played: "<<matches<<endl;;

  }

  public:
```

```
  cricketer(string name,string dob,int
        matches):name(name),dob(dob),matches(matches)

  {/*cout<<"cricketer"<<endl;*/}

  string get_name() const {return name;}

  string get_dob() const {return dob;}

  int get_matches() const {return matches;}

  void set_matches(int m) {matches = m;}

  virtual void print_details() const = 0;

};


class batsman: virtual public cricketer{

  int runs;

  float avg;

  public:

  batsman(string name,string dob,int matches,int runs,float
        avg):cricketer(name,dob,matches),runs(runs),avg(avg){/
        *cout<<"batsman"<<endl;*/}


  int get_runs() const {return runs;}

  float get_avg_score() const {return avg;}

  void set_runs(int r) {runs = r;}

  void set_avg_score(float a) {avg = a;}


  void print_details() const{
```

```
    print_basic_details();
    cout<<"Batting:"<<endl;
    cout<<"Runs: "<<runs<<endl;
    cout<<"Average score: "<<avg<<endl;
    cout<<"----------------------"<<endl;
  }
};

class bowler: virtual public cricketer{
  int wickets;
  float avg;
  public:
  bowler(string name,string dob,int matches,int wickets,float
        avg):cricketer(name,dob,matches),wickets(wickets),avg(avg){/
        *cout<<"bowler"<<endl;*/}

  int get_wickets() const {return wickets;}
  float get_avg_economy() const {return avg;}
  void set_wickets(int w) {wickets = w;}
  void set_avg_economy(float a) {avg = a;}

  void print_details() const{
    print_basic_details();
```

```
    cout<<"Bowling:"<<endl;
    cout<<"Wickets taken: "<<wickets<<endl;
    cout<<"Average economy: "<<avg<<endl;
    cout<<"----------------------"<<endl;
  }
};

class all_rounder : public batsman,public bowler{
  public:
  all_rounder(string name,string dob,int matches,int runs,float avg_sc,int
        wickets,float
        avg_ec):batsman(name,dob,matches,runs,avg_sc),bowler(name,dob,
        matches,wickets,avg_ec),cricketer(name,dob,matches){/
        *cout<<"all_rounder"<<endl;*/}
  void print_details() const{
    cout<<"All Rounder"<<endl;
    batsman::print_details();
    bowler::print_details();
    cout<<"----------------------"<<endl;
  }
};

class dw_pair{
  batsman *bat;
```

```
bowler *bowl;
public:
dw_pair(string name1,string dob1,int matches1,string name2,string
      dob2,int matches2,int runs,float avg_sc,int wickets,float avg_ec){
  //cout<<"dw_pair"<<endl;
  bat = new batsman(name1,dob1,matches1,runs,avg_sc);
  bowl = new bowler(name2,dob2,matches2,wickets,avg_ec);
}

int get_runs() const {return bat->get_runs();}
float get_avg_score() const {return bat->get_avg_score();}
void set_runs(int r) {bat->set_runs(r);}
void set_avg_score(float a) {bat->set_avg_score(a);}

int get_wickets() const {return bowl->get_wickets();}
float get_avg_economy() const {return bowl->get_avg_economy();}
void set_wickets(int w) {bowl->set_wickets(w);}
void set_avg_economy(float a) {bowl->set_avg_economy(a);}

void print_details() const{
   cout<<"Double Wicket Pair"<<endl;
   bat->print_details();
   bowl->print_details();
```

```
  }

  ~dw_pair(){delete bat; delete bowl;}
};

int main()
{
 batsman bat("bat1","1/1/1",100,100,50.5);
 bowler bowl("bowl1","2/2/2",90,90,40.5);
 all_rounder ar("AR","3/3/3",110,110,50,30,15);
 dw_pair dw("Dw1","4/4/4",13,"Dw2","5/5/5",15,10,5.3,4,150);

 bat.print_details();
 bowl.print_details();
 ar.print_details();
 dw.print_details();
 return 0;
}
```

```
Name: bat1
DOB: 1/1/1
Total matches played: 100
Batting:
Runs: 100
Average score: 50.5
----------------------
Name: bowl1
DOB: 2/2/2
Total matches played: 90
Bowling:
Wickets taken: 90
Average economy: 40.5
----------------------
All Rounder
Name: AR
DOB: 3/3/3
Total matches played: 110
Batting:
Runs: 110
Average score: 50
----------------------
Name: AR
DOB: 3/3/3
Total matches played: 110
Bowling:
Wickets taken: 30
Average economy: 15
----------------------
----------------------
Double Wicket Pair
Name: Dw1
DOB: 4/4/4
Total matches played: 13
Batting:
Runs: 10
Average score: 5.3
----------------------

Name: Dw2
DOB: 5/5/5
Total matches played: 15
Bowling:
Wickets taken: 4
Average economy: 150
----------------------
```