```
enum AccountType{
Savings = 1,
Current,
Dmat
}
```

```
class InSufficientFundsException{
string message;
InSufficientFundsException(){
}
InSufficientFundsException(string message){
this->message = message
}
void display(){
cout<<mesage<<endl;
}
}
```

```
class BankAccount{
AccountType type;

accept(){
cout<<"1. Savings"<<endl;
cout<<"2. Current"<<endl;
cout<<"3. Dmat"<<endl;
cout<<"Enter the choice of account<<endl;
int choice;
cin>>choice;
type = AccountType(choice);
}

display(){

switch(type)
case Savings:
cout<<"Account Type = Savings<<endl;
break;
case Current:
cout<<"Account Type = Current<<endl;
break;
case Dmat:
cout<<"Account Type = Dmat<<endl;
break;
}
}
```

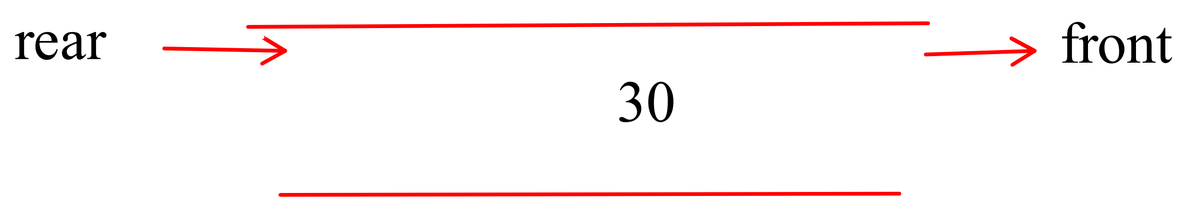
```
deposit(int amount){
        if(amount<0)
                throw InSufficientFundsException
                        ("deposit amt cannot be -ve)
}

withdraw(int amount){
        if(amount>balance)
                throw InSufficientFundsException
                        ("amt cannot be > then balance
}
```

Person
Employee
Student

mock

Lab
1. Solve tthe bank account question
2. Solve the yesterday Student remaining part from the class demo
3. Do the file IO / classwork practice
4. Solve the Mock Paper

vector



Abstraction
Encapsulation
Modularity
Hireracy
Polymorphism
Concurrency
Persistance

3000*20

$$\begin{tikzcd}
 & & \text{ios} & & \\
 & \text{istream} \arrow[ur] & & \text{ostream} \arrow[ul] & \\
 & & \text{iostream} \arrow[ul] \arrow[ur] & & \\
 & \text{ifstream} \arrow[uu] & & \text{ofstream} \arrow[uu] & \\
 & & \text{fstream} \arrow[u] & &
\end{tikzcd}$$

cin,
cout

1 10000 ss -> 1, Anil,10000
↑

Anil

Person{

}
Employee{

}

Student{

}

Program start
    1. Read the files
    2. Add the data in to vectors
    vector<Employee *>
    vector<Student *>
    3. Changes to be done in the vector
Before Program ends
    4. Write the entire vectors into the
    respective files

employee.txt

student.txt

```
Person *p = new Employee(); // UPCASTING
p->display();

Employee * e = (Employee *)p;


vector<Person *> v1;
v1.push_back(p);

vector<Employee *> v2;
v2.push_back(p); // NOT OK
```

```
namespace NEmployee{
class Employee:public Person{

static void loadEmployees();
}


void saveEmployees();
int findEmployee();
}
```

```
using namespace NEmployee;
loadEmployees();

Employee::loadEmployees();
```

```
class Person{
name
}

class Customer : public Person{
cid,
mobile
vector<Product *> purchasedproducts;

void dopurchase(productlist){
//display product list
// enter product id to purchase
// search if product with given product id exists
// purchased_product.push_back(productlist[i]);
}
}
```

```
class Product{
id,
name,
price
}
```

```
vector<Product*> productlist;
vector<Customer *> customerlist;

int index = findcustomer(){
customerlist[index]-> dopurchase(productlist);
}
```