

```

1 #include "Account.h"
2 #include "SavingsAcc.h"
3 #include "CheckingAcc.h"
4 #include <iostream>
5 #include <vector>
6 #include <typeinfo>
7 using namespace std;
8
9 int main() {
10     vector<Account*> accounts;
11     double withdraw = 0.0;
12     double deposit = 0.0;
13
14     // Create SavingsAccount and CheckingAccount objects
15     accounts.push_back(new SavingsAccount(500.0, 0.05)); // Savings account with $500 initial balance and 5% interest rate
16     accounts.push_back(new CheckingAccount(200.0, 1.0)); // Checking account with $200 initial balance and $1 fee per transaction
17
18     // Process each account
19
20     for (Account *account : accounts){ // pointer "account" 遍及 accounts(可知每格為何種object)
21
22         SavingsAccount *savings = dynamic_cast <SavingsAccount*> (account);
23         CheckingAccount *checking = dynamic_cast <CheckingAccount*> (account);
24
25         if(savings != 0){
26             cout << "Processing SavingsAccount account." << endl;
27
28             cout << "Enter amount to deposit: ";
29             cin >> deposit;
30             account -> credit(deposit);
31
32             cout << "Enter amount to withdraw: ";
33             cin >> withdraw;
34             account -> debit(withdraw);
35
36             double interest = savings -> calculateInterest();
37             cout << "Interest added: " << interest << endl;
38             cout << "Updated balance: " << savings -> getBalance() + interest << endl << endl;
39         }

```

```

40         else if (checking != 0){
41             cout << "Processing CheckingAccount account." << endl;
42
43             cout << "Enter amount to deposit: ";
44             cin >> deposit;
45             account -> credit(deposit);
46
47             cout << "Enter amount to withdraw: ";
48             cin >> withdraw;
49             account -> debit(withdraw);
50
51             cout << "Updated balance: " << checking -> getBalance() << endl << endl;
52         }
53     }
54
55     // Clean up dynamically allocated memory
56
57     for (Account* account : accounts){
58         delete account;
59     }
60
61     return 0;
62 }
63
64

```

```

1  #ifndef ACCOUNT_H
2  #define ACCOUNT_H
3
4  class Account{
5      public:
6          Account();
7          Account(double);
8
9          virtual void credit(double);
10         virtual bool debit(double);
11
12         void setBalance(double);
13         double getBalance() const;
14
15     private:
16         double balance;
17 };
18

```

```

1  #include "Account.h"
2  #include <iostream>
3  using namespace std;
4
5  Account::Account(){}
6
7  Account::Account(double ba) : balance(0.0){
8      setBalance(ba);
9  }
10
11 void Account::credit(double cr){
12     balance += cr;
13 }
14
15 bool Account::debit(double deb){
16     if(deb > balance){
17         cout << "Debit amount exceeded account balance" << endl;
18         return false;
19     }
20     else{
21         balance -= deb;
22         return true;
23     }
24 }
25
26 void Account::setBalance(double ba){
27     balance = ba;
28 }
29
30 double Account::getBalance() const{
31     return balance;
32 }
33

```

```

1  #ifndef SAVINGSACCOUNT_H
2  #define SAVINGSACCOUNT_H
3
4  #include "Account.h"
5
6  class SavingsAccount : public Account{
7      public:
8          SavingsAccount();
9          SavingsAccount(double, double);
10
11         double calculateInterest() const;
12
13         void setInterestRate(double);
14         double getInterestRate() const;
15
16     private:
17         double interest_rate;
18
19
20 };
21
22 #endif
23

```

```

1  #include "SavingsAcc.h"
2  #include <iostream>
3  using namespace std;
4
5  SavingsAccount::SavingsAccount(){}
6
7  SavingsAccount::SavingsAccount(double ba, double rate) : interest_rate(0.0){
8      Account::setBalance(ba);
9      setInterestRate(rate);
10 }
11
12 double SavingsAccount::calculateInterest() const{
13     return (Account::getBalance() * interest_rate);
14 }
15
16 void SavingsAccount::setInterestRate(double rate){
17     if (rate < 0.0)
18     {
19         cout << "The initial interest rate is invalid." << endl;
20     }
21     else
22     {
23         interest_rate = rate;
24     }
25 }
26
27 double SavingsAccount::getInterestRate() const{
28     return interest_rate;
29 }
30

```

```
1  #ifndef CHECKINGACCOUNT_H
2  #define CHECKINGACCOUNT_H
3
4  #include "Account.h"
5
6  class CheckingAccount : public Account{
7      public:
8          CheckingAccount();
9          CheckingAccount(double, double);
10
11          virtual void credit(double);
12          virtual bool debit(double);
13
14          void setTransactionFee(double);
15          double getTransactionFee();
16
17      private:
18          double transaction_fee;
19
20  };
21
22  #endif
```

```

1 #include "CheckingAcc.h"
2 #include <iostream>
3 using namespace std;
4
5 CheckingAccount::CheckingAccount(){}
6
7 CheckingAccount::CheckingAccount(double ba, double fee) : transaction_fee(0.0){
8     Account::setBalance(ba);
9     setTransactionFee(fee);
10 }
11
12 void CheckingAccount::credit(double cr){
13     Account::credit(cr - transaction_fee);
14 }
15
16 bool CheckingAccount::debit(double deb){
17     if (Account::debit(deb + transaction_fee)){
18         return true;
19     }
20     else{
21         cout << "Debit amount exceeded account balance" << endl;
22         return false;
23     }
24 }
25
26 void CheckingAccount::setTransactionFee(double fee){
27     if (fee < 0.0){
28         cout << "The initial transaction fee is invalid." << endl;
29     }
30     else{
31         transaction_fee = fee;
32     }
33 }
34
35 double CheckingAccount::getTransactionFee(){
36     return transaction_fee;
37 }

```

Processing SavingsAccount account.

Enter amount to deposit: 100

Enter amount to withdraw: 50

Interest added: 27.5

Updated balance: 577.5

Processing CheckingAccount account.

Enter amount to deposit: 100

Enter amount to withdraw: 50

Updated balance: 248