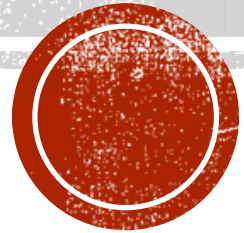
THE BANK APPLICATION.



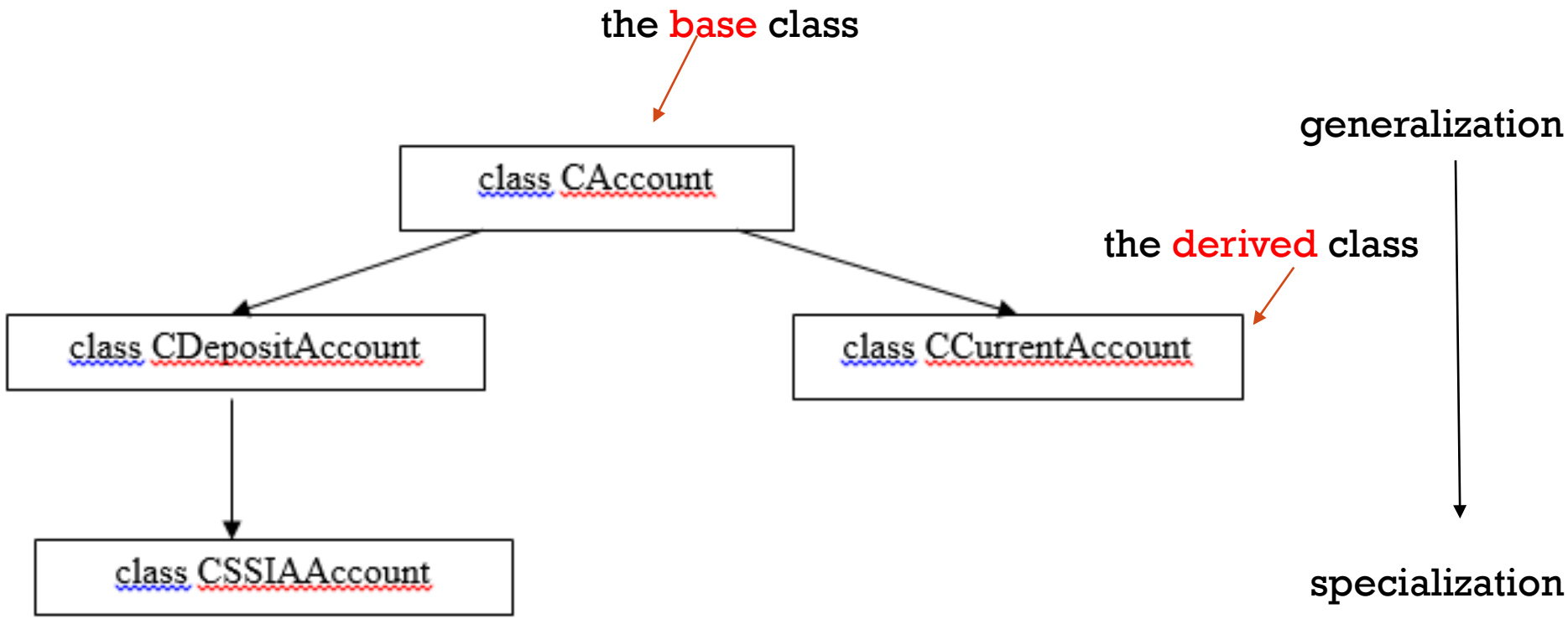
THE BANK RULES

- The bank has a number of accounts:
- Accounts in the Bank are either Current, or Deposit accounts
- Deposit accounts get payed interest payments on their balance – and hence have an interest rate associated with them.
- It is possible to have an overdraft on a current account i.e. you can withdraw money to the overdraft limit – and hence have an overdraft limit associated with them
- An SSIA account is a type of deposit account – with two addition rules
  - 1. Withdrawals are not permitted
  - 2. Every lodgement has an addition 25% added to it.
- The data in the bank will have no persistence i.e. when the application is ended all data in the program will be lost

BANKING APPLICATION

- To your Banking application add the following menu options:

- Option 1: Create account
- Option 2: Show All Account Details.
- Option 3: Lodge.
- Option 4: Withdraw.
- Option 5: Maintance: Run quarter interest rate
- Option 6: Exit


the base class
class CAccount
generalization
the derived class
class CDepositAccount
class CCurrentAccount
class CSSIAAccount
specialization

```
public abstract class Account
{
    5 references
    public double balance { get; set; }
    3 references
    public string id { get; set; }

    1 reference
    public Account(double balance, string id)
    {
        this.balance = balance;
        this.id = id;
    }

    1 reference
    public Account()
    {
        balance = 0;
        id = "";
    }

    1 reference
    public Account(string id)
    {
        balance = 0;
        this.id = id;
    }


    1 reference
    public virtual void Lodge(double amount)
    {
        balance += amount;
    }
    2 references
    public abstract void Withdraw(double amount);
    3 references
    public abstract void ShowDetails();
}
```

For the banking hierarchy the **class CAccount** will act as an abstract base class.

The constructor functions will appropriately initialize the data members.

Note : virtual – expected to be over written ..but does not have to

Note : abstract – implementation details will have to be given in the derived class*
And Account has to be an abstract class

BANKING EXAMPLE

- However the **Lodge( )** function, whilst defined and implemented in this class, is specified as virtual. This means that it is likely to be redefined in some of the other classes of the hierarchy. The **Lodge( )** function will need to be redefined in the **class CSSIAAccount** as for every lodgment an additional 25% of the amount must be added to the balance.

- The **ShowDetails( )** and **Withdraw( )** functions are undefined as abstract methods. This means that the class is an abstract base class. The derived classes must give implementation details of the abstract method or become an abstract class itself.

DERIVATION HIERARCHY

- The class CAccount is an abstraction…
  - It represents an encapsulation of common
    - Properties
    - Attributes
- Its sole purpose
  - Define a class from which to derive other classes….
    - It encapsulates common
      - Data members
      - Function members
- Called
  - Abstract Base Class
  - Abstract Super Class

ABSTRACT BASE CLASSES

- An abstract base class is not instantiated, but other classes are derived from it.
- A virtual member function in a base class **expects** to be overridden in a derived class
- An abstract member function of a base class **must** be overridden.

CLASS DERIVATION

Any class can serve as a base class…

- Thus a derived class can also be a base class

- Worth spending time at the outset of a design to develop sound definitions

- We can now add a derived classes to the hierarchy. Firstly we can add the class representing current account type accounts. These type of accounts have credit limits but do not attract interest.

To add the **class CCurrentAccount** to the hierarchy specify that the class is derived from **class CAccount.**

```
public class CurrentAccount : Account
{
    4 references
    public int creditLimit {get; set; }

    1 reference
    public CurrentAccount(string id):base(id)
    {
        creditLimit = 0;
    }

    0 references
    public CurrentAccount():base()
    {
        creditLimit = 0; ;
    }

    0 references
    public CurrentAccount(double balance, string id, double creditLimit):base(balance, id)
    {
        creditLimit = 0;
    }

    2 references
    public override void Withdraw(double amount)
    {
        if (amount <= balance + creditLimit)
            balance -= amount;
        else
            Console.WriteLine("Insufficient funds");
    }

    3 references
    public override void ShowDetails()
    {
        Console.WriteLine("{0}   {1,10}  {2,10}", id, balance.ToString("F2"),creditLimit.ToString("F2"));
    }
}
```

In most cases the derivation is specified as **public**. The use of public means that the public/private/protected access permissions continue down the hierarchy

BANKING EXAMPLE

- In adding data members to the class, only specify data members that are additional to those that will be inherited from the base class. In this case the class should specify a data member **creditLimit** .

- Functions defined and implemented in the base class that are suitable for use with objects of this class are not redefined e.g. **get/set methods**

- In general, functions in the base class that are undefined must be defined in the derived class. The exception to this is if the derived class itself is to be an abstract base class.

- Additional member functions may be added to the class, to handle the increased specialization of this class. Typically these functions will be required because of the additional data members e,g, **set and get credit limit.**

BANKING EXAMPLE

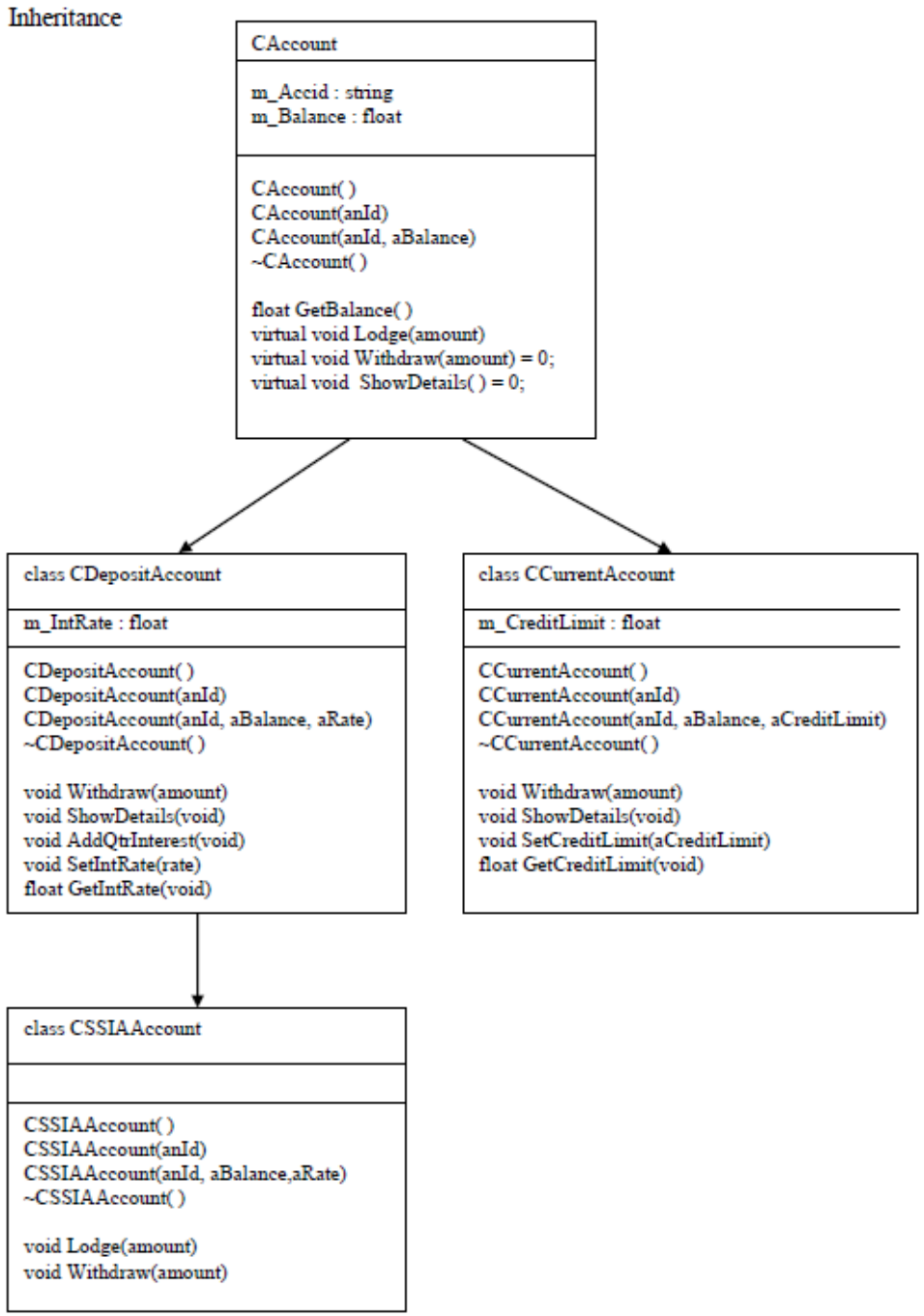
- When an object of this class is instantiated, before the class constructor function is called, the constructor for the base class is called.
- If the derived classes constructor is passed arguments, some of these may need to be passed on to the base class constructor as shown, using a 'member initialization list'.
- The derived class constructor need only initialize the additional data member, since the inherited data members will have been initialized by the base class constructor.
- The derived class constructor may initialize the additional data member using the member initialization list or may use assignment statements in the body of the function

```
3 references
class Deposit:Account
{
    5 references
    public double IntRate {get;set;}

    0 references
    public Deposit():base()
    {
        IntRate= 0;
    }
    0 references
    public Deposit(string id, double intr):base(id)
    {
        IntRate= intr;
    }
    0 references
    public Deposit(double balance, string id, double intr):base(balance, id)
    {
        IntRate = intr;
    }

    0 references
    void withdraw(float amount)
    {
        if (amount <= balance)

            balance -= amount;

        else

            Console.Write("not allowed as amount > balance");
       }
    4 references
    public override void ShowDetails()
    {
        Console.WriteLine("{0}   {1,10}  {2,10}", id, balance.ToString("F2"), IntRate.ToString("F2"));
    }
    0 references
    public void AddQtrInterest(){
        balance = balance* IntRate / 100 /4;
    }
}
```

BANKING EXAMPLE

- Next we add the final class representing SSIA type accounts. These type accounts represent deposit type accounts but each lodged amount has 25% added to the amount by the government. However, direct withdrawals may not be made from these type accounts. To withdraw the money the account must be closed.

UML CLASS DIAGRAM



SSIA ACCOUNT

- ??? Over to you
- Remember –
  - no with withdrawals from the account
  - And 25% extra on lodgements