DevStyleR ACADEMY

Programming with C++ Core

OBJECT-ORIENTED PROGRAMMING: INHERITANCE

2024

DevStyleR

Summary

- Base Classes and Derived Classes
- Protected Members
- Relationship between Base Classes and Derived Classes
    - Creating a CommissionEmployee - BasePlusCommissionEmployee Inheritance Hierarchy Using private Data
- Constructors and Destructors in Derived Classes
- Public, protected and private Inheritance

.....

DevStyleR ACADEMY

BASE CLASSES AND DERIVED CLASSES

- Base classes and derived classes
  - Object of one class “is an” object of another class
    - Example: Rectangle is quadrilateral
      - Class Rectangle inherits from class Quadrilateral
        - Quadrilateral is the base class
        - Rectangle is the derived class
  - Base class typically represents larger set of objects than derived classes
    - Example:
      - Base class: Vehicle
        - Includes cars, trucks, boats, bicycles, etc.
      - Derived class: Car
        - Smaller, more-specific subset of vehicles

DevStyleR ACADEMY

Base Class	Derived Classes
Student	Graduate Student, Undergraduate Student
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	Car Loan, Home Improvement Loan, Mortgage Loan
Employee	Faculty, Staff
Account	Checking Account, Savings Account

DevStyleR ACADEMY

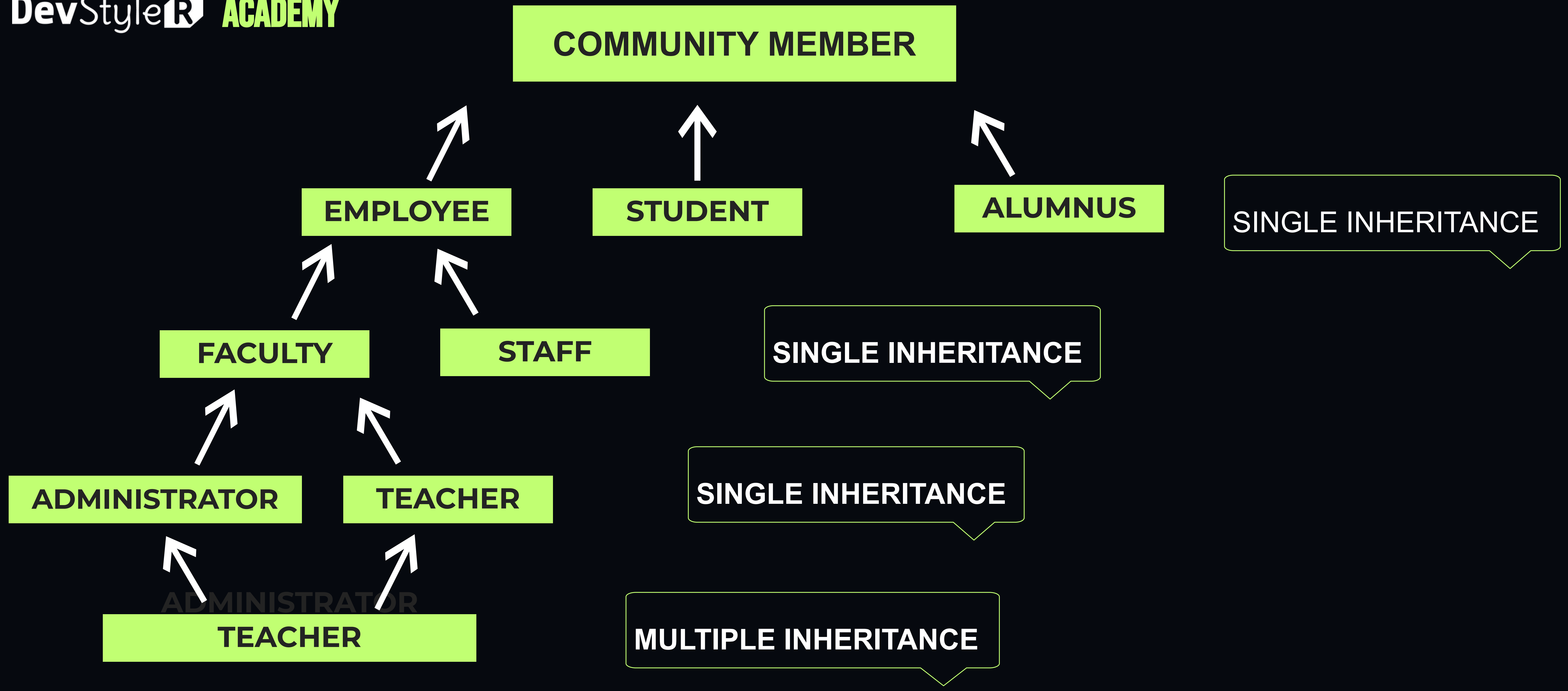
BASE CLASSES AND DERIVED CLASSES

- Inheritance hierarchy
  - Inheritance relationships: tree-like hierarchy structure
  - Each class becomes
    - Base class
      - Supplies data/behaviors to other classes

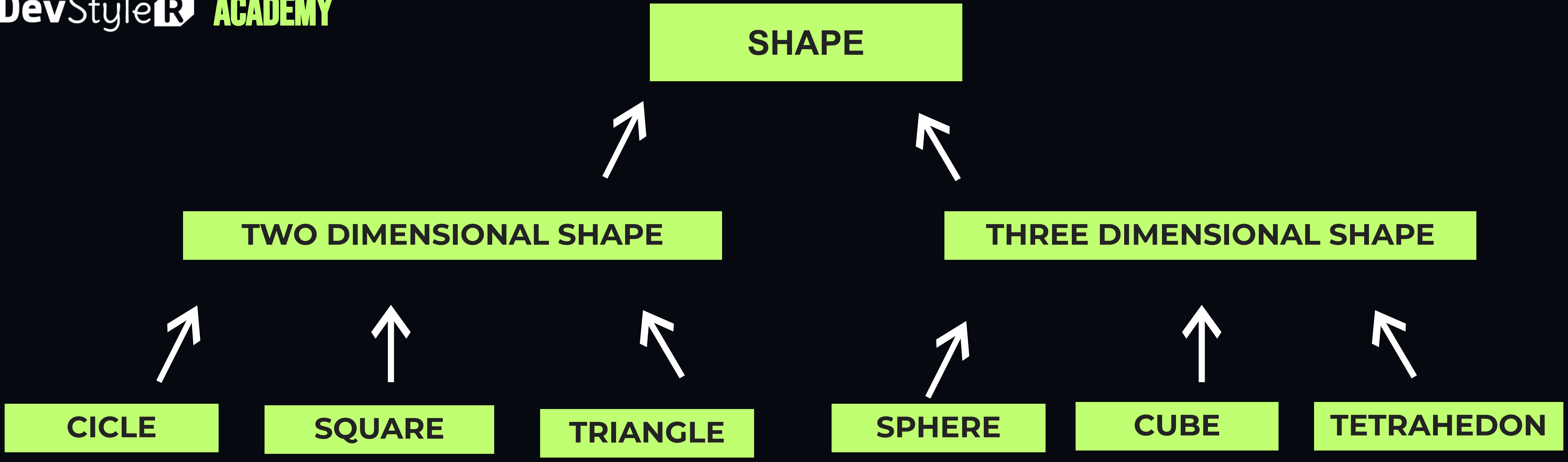
    Or

    - Derived class
      - Inherits data/behaviors from other classes

DevStyleR ACADEMY



Inheritance hierarchy for university CommunityMembers



Inheritance hierarchy for Shapes

DevStyleR ACADEMY

BASE CLASSES AND DERIVED CLASSES

- public inheritance
  - Specify with:

    Class *TwoDimensionalShape* : public *Shape*
    - Class TwoDimensionalShape inherits from class Shape
  - Base class private members
    - Not accessible directly
    - Still inherited
      - Manipulated through inherited public member functions
  - Base class public and protected members
    - Inherited with original member access
  - friend functions
    - Not inherited

DevStyleR ACADEMY

PROTECTED MEMBERS

- protected access
  - Intermediate level of protection between public and private
  - protected members are accessible to
    - Base class members
    - Base class friends
    - Derived class members
    - Derived class friends
- Derived-class members
  - Refer to public and protected members of base class
    - Simply use member names
  - Redefined base class members can be accessed by using base-class name and binary scope resolution operator (::)

DevStyleR ACADEMY

RELATIONS BETWEEN BASE CLASSES AND DERIVED CLASSES

- Base class and derived class relationship
  - Example: CommissionEmployee/BasePlusCommissionEmployee inheritance hierarchy
    - CommissionEmployee
      - First name, last name, SSN, commission rate, gross sale amount
    - BasePlusCommissionEmployee
      - First name, last name, SSN, commission rate, gross sale amount
      - And also: base salary

COMMISSIONEMPLOYEE - BASEPLUSCOMMISSIONEMPLOYEE INHERITANCE HIERARCHY USING PRIVATE DATA

- The hierarchy
  - Use the best software engineering practice
    - Declare data members as private
    - Provide public *get* and *set* functions
    - Use *get* method to obtain values of data members

DevStyleR ACADEMY

Programming with C++ Core

LIVE DEMO

CommissionBaseEmployee

2024

CONSTRUCTORS AND DESTRUCTORS IN DERIVED CLASSES

- Instantiating derived-class object
  - Chain of constructor calls
    - Derived-class constructor invokes base class constructor
      - Implicitly or explicitly
    - Base of inheritance hierarchy
      - Last constructor called in chain
      - First constructor body to finish executing
      - Example: CommissionEmployee/BasePlusCommissionEmployee hierarchy
        - CommissionEmployee constructor called last
        - CommissionEmployee constructor body finishes execution first
    - Initializing data members
      - Each base-class constructor initializes its data members that are inherited by derived class

DevStyleR ACADEMY

CONSTRUCTORS AND DESTRUCTORS IN DERIVED CLASSES

- Destroying derived-class object
  - Chain of destructor calls
    - Reverse order of constructor chain
    - Destructor of derived-class called first
    - Destructor of next base class up hierarchy next
      - Continue up hierarchy until final base reached
        - After final base-class destructor, object removed from memory
- Base-class constructors, destructors, assignment operators
  - Not inherited by derived classes

PUBLIC, PROTECTED AND PRIVATE INHERITANCE

- public inheritance
  - Base class public members derived class public members
  - Base class protected members derived class protected members
  - Base class private members are not accessible
- protected inheritance (not *is-a* relationship)
  - Base class public and protected members derived class protected members
- private inheritance (not *is-a* relationship)
  - Base class public and protected members derived class private members

DevStyleR ACADEMY

BASE-CLASS MEMBER ACCESS SPECIFIER	TYPE OF INHERITANCE		
	PUBLIC INHERITANCE	PROTECTED INHERITANCE	PRIVATE iNHERITANCE
PUBLIC	PUBLIC IN DERIVED CLASS Can be accessed directly by member functions, friend functions, and nonmember functions.	PROTECTED IN DERIVED CLASS Can be accessed directly by member functions and friend functions.	PRIVATE IN DERIVED CLASS Can be accessed directly by member functions and friend functions.
PROTECTED	PROTECTED IN DERIVED CLASS Can be accessed directly by member functions and friend functions.	PROTECTED IN DERIVED CLASS Can be accessed directly by member functions and friend functions.	PRIVATE IN DERIVED CLASS Can be accessed directly by member functions and friend functions.
PRIVATE	HIDDEN IN DERIVED CLASS Can be accessed directly by member functions and friend functions through public or protected member functions of the base class.	HIDDEN IN DERIVED CLASS Can be accessed directly by member functions and friend functions through public or protected member functions of the base class.	HIDDEN IN DERIVED CLASS Can be accessed directly by member functions and friend functions through public or protected member functions of the base class.

DevStyleR ACADEMY

THANK YOU

ACADEMY@DEVSTYLER.IO

DevStyleR ACADEMY

Programming with C++ Core

EXERCISES

2024

TASK 1

Many programs written with inheritance could be written with composition instead, and vice versa.

- Rewrite class *BasePlusCommissionEmployee* of the *CommissionEmployeeBasePlusCommissionEmployee* hierarchy to use composition rather than inheritance.
- After you do this, assess the relative merits of the two approaches for designing classes *CommissionEmployee* and *BasePlusCommissionEmployee*, as well as for object-oriented programs in general.

TASK 2

Draw an inheritance hierarchy for students at a university similar to the hierarchy shown in Slide 6.

- Use *Student* as the base class of the hierarchy, then include classes *UndergraduateStudent* and *GraduateStudent* that derive from *Student*.
- Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, *Freshman*, *Sophomore*, *Junior* and *Senior* might derive from *UndergraduateStudent*, and *DoctoralStudent* and *MastersStudent* might derive from *GraduateStudent*.
- After drawing the hierarchy, discuss the relationships that exist between the classes.

2.

TASK 3

The world of shapes is much richer than the shapes included in the inheritance hierarchy of Slide 7.

- Write down all the shapes you can think of both two-dimensional and three-dimensional and form them into a more complete *Shape* hierarchy with as many levels as possible.

Your hierarchy should have base class Shape from which class *TwoDimensionalShape* and class *ThreeDimensionalShape* are derived.

TASK 4

Draw an inheritance hierarchy for classes *Quadrilateral*, *Trapezoid*, *Parallelogram*, *Rectangle* and *Square*.

- Use *Quadrilateral* as the base class of the hierarchy.
- Make the hierarchy as deep as possible.

TASK 5

Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated.

- Create an inheritance hierarchy to represent various types of packages.
- Use *Package* as the base class of the hierarchy, then include classes *TwoDayPackage* and *OvernightPackage* that derive from *Package*.
- Base class *Package* should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package.
- *Package*'s constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values. *Package* should provide a public member function *calculateCost* that returns a double indicating the cost associated with shipping the package.
- *Package*'s *calculateCost* function should determine the cost by multiplying the weight by the cost per ounce.

TASK 5

DevStyleR

- Derived class *TwoDayPackage* should inherit the functionality of base class *Package*, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service.
- *TwoDayPackage*'s constructor should receive a value to initialize this data member. *TwoDayPackage* should redefine member function *calculateCost* so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class *Package*'s *calculateCost* function.
- Class *OvernightPackage* should inherit directly from class *Package* and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service.
- *OvernightPackage* should redefine member function *calculateCost* so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost.
- Write a test program that creates objects of each type of *Package* and tests member function *calculateCost*.

TASK 6

Create an inheritance hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts.

More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold.
Savings accounts, for instance, earn interest on the money they hold. *Checking* accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

- Create an inheritance hierarchy containing base class *Account* and derived classes *SavingsAccount* and *CheckingAccount* that inherit from class *Account*.
- Base class *Account* should include one data member of type double to represent the account balance.

TASK 6

DevStyleR

- The class should provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it is greater than or equal to 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid.
- The class should provide three member functions.
- Member function credit should add an amount to the current balance. Member function debit should withdraw money from the Account and ensure that the debit amount does not exceed the Account's balance.
- If it does, the balance should be left unchanged and the function should print the message "Debit amount exceeded account balance." Member function getBalance should return the current balance.

TASK 6

- Derived class *SavingsAccount* should inherit the functionality of an *Account*, but also include a data member of type double indicating the interest rate (percentage) assigned to the *Account*. *SavingsAccount*'s constructor should receive the initial balance, as well as an initial value for the *SavingsAccount*'s interest rate.
- *SavingsAccount* should provide a public member function *calculateInterest* that returns a double indicating the amount of interest earned by an account.
- Member function *calculateInterest* should determine this amount by multiplying the interest rate by the account balance. [Note: *SavingsAccount* should inherit member functions credit and debit as is without redefining them.]
- Derived class *CheckingAccount* should inherit from base class *Account* and include an additional data member of type double that represents the fee charged per transaction.

TASK 6

DevStyleR

- *CheckingAccount*'s constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class *CheckingAccount* should redefine member functions *credit* and *debit* so that they subtract the fee from the account balance whenever either transaction is performed successfully.
- *CheckingAccount*'s versions of these functions should invoke the base-class *Account* version to perform the updates to an account balance.
- *CheckingAccount*'s debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance).
- [Hint: Define *Account*'s debit function so that it returns a bool indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]
- After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the *SavingsAccount* object by first invoking its *calculateInterest* function, then passing the returned interest amount to the object's *credit* function.