2.1 Klassen-Deklaration

Auf der nächsten Seite ist eine lückenhafte Deklaration einer Klasse `BankAccount` gegeben, die ein Bankkonto mit einem Kontostand repräsentiert. Der Code liegt auch als entsprechende Header-Datei namens `BankAccount.h` vor. Fülle die Deklaration anhand der Anforderungen der folgenden Teilaufgaben aus und implementiere die Definition in einer entsprechenden `BankAccount.cpp`, sodass sich die in der Datei `Exercise2_1.cpp` gegebene `main`-Funktion korrekt ausführen lässt.

a) Die Klasse soll den **Namen des Kontoinhabers**, den **Kontostand** und eine geheime **Sicherheitsnummer** speichern und für alle Attribute, außer der Sicherheitsnummer, Getter zur Verfügung stellen.

b) Es soll ein Konstruktor existieren, mit dem alle Attribute anhand der übergebenen Parameter gesetzt werden. Außerdem stellt die Klasse einen Destruktor zur Verfügung.

c) Zum Einzahlen von Geld soll eine Methode namens `deposit` existieren und zum Abheben von Geld eine Methode namens `withdraw`.

d) Mithilfe einer statischen Variable soll gezählt werden, wie viele Bankkonten zur Laufzeit existieren. Die Variable soll nur über einen Getter von außerhalb der Klasse zugänglich sein.

e) Eine Nicht-Member-Funktion namens `checkSecretCode` erhält eine konstante Referenz auf ein Bankkonto und eine Nummer, die mit der geheimen Sicherheitsnummer des übergebenen Kontos verglichen wird.

f) Eine externe Funktion soll die Summe der Kontostände zweier Bankkonten berechnen.

```cpp
#pragma once
#include <iostream>
#include <string>

class BankAccount
{
// TEILAUFGABE A
--------------:
    ----------  ----------------;    // Name des Kontoinhabers
    ----------  ----------------;    // Kontostand
    ----------  ----------------;    // Geheimer Code des Kontos

--------------:
    /// Getter fuer den Kontoinhaber
    -------------  ----------------------()  -----------;

    /// Getter fuer den Kontostand
    -------------  ----------------------()  -----------;

// TEILAUFGABE B
--------------:
    /// Konstruktor mit Parametern
    --------------(------------------ accountHolder, ------------ initialBalance,
                   ------------ secretCode);

    /// Destruktor
    ----------------------();

// TEILAUFGABE C
--------------:
    /// Methode, um Geld einzuzahlen
    ------------- -------------(-------------- --------------);

    /// Methode, um Geld abzuheben
    ------------- -------------(-------------- --------------);

// TEILAUFGABE D
--------------:
    ------------- ------------- s_accountCount;  // Statische Variable zur Zaehlung
        der Bankkonten

--------------:
    /// Gibt die Anzahl aller Bankkonten zurueck
    ------------- ------------- ----------------------();

// TEILAUFGABE E
public:
    /// Freundfunktion: Ueberprueft den geheimen Code eines Kontos
    ------------- ------------- -------------(------------------- account,
                                              --------- code);
};

// TEILAUFGABE F
/// Berechnet die Summe der Guthaben zweier Konten
------------- ------------- calculateTotalBalance(-------------------- acc1,
                                                  -------------------- acc2)
{
    return acc1.getBalance() + acc2.getBalance();
}
```

2.2 Eine „Vector“-Klasse

In dieser Aufgabe soll eine Klasse `Vector` implementiert werden, in der mehrere Elemente vom Datentyp `double` gespeichert werden können. Die Elemente sollen in einem Array gespeichert werden und die Klasse soll Größe des Arrays verwalten, sodass auch während der Laufzeit eine Änderung der Größe ermöglicht wird.

Die einzelnen Teilaufgaben führen Dich dabei Schrittweise durch die Erstellung der Klasse hindurch und fügen der Klasse weitere Funktionalitäten hinzu. Schreibe für alle implementierten Funktionalitäten passende Anwendungsbeispiele.

a) Erstelle einen Standardkonstruktor, der die Größe des Arrays auf 0 festlegt und noch keinen Speicherplatz allokiert. Zusätzlich soll ein Konstruktor vorhanden sein, mit dem die Größe des Arrays initial festgelegt werden kann.

b) Erstelle einen Destruktor, der den allokierten Speicherplatz wieder freigibt.

c) Erstelle Getter und Setter für die Größe des Arrays und sorge dafür, dass bei einer Änderung der Größe neuer Speicherplatz allokiert, die gespeicherten Elemente an den neuen Speicherplatz kopiert und der alte Speicherplatz freigegeben wird.

d) Überschreibe den Indizierungs-Operator `[]`, um Zugriff auf Elemente im Array zu ermöglichen. Dabei soll nicht überprüft werden, ob der angegebene Index im Bereich vom Array liegt.

e) Die Funktion `at()` soll dieselbe Funktionalität wie der Indizierungs-Operator ermöglichen und zusätzlich überprüfen, ob der angegebene Index gültig ist. Sollte der Index außerhalb des Bereiches liegen, soll das erste Element des Arrays zurückgegeben werden. Überlege dir Vor- und Nachteile für ein derartiges Vorgehen.

f) Überlege dir eine beliebige Möglichkeit, wie zwei Instanzen der Klasse miteinander verglichen werden und in eine Reihenfolge gebracht werden können. Überlade die Vergleichsoperatoren für deine Klasse entsprechend deinen Überlegungen.

g) Überlade den Output- und Input-Operator, um eine Ausgabe der Elemente des Array und eine Eingabe über die Konsole zu ermöglichen. Was soll passieren, wenn bei der Eingabe mehr/weniger Elemente eingegeben werden als im Array bisher gespeichert sind? Implementiere in diesem Fall ein Verhalten deiner Wahl und füge der Funktion Kommentare hinzu, die dieses Verhalten einem möglichen Nutzer erläutern.

h) Erstelle eine Funktion `push_back()`, die dem Nutzer ermöglicht, ein einzelnes Element dem Array an dessen Ende hinzuzufügen und damit die Größe des Arrays um eins zu erhöhen.

i) Überlade die Operatoren `+`, `-` und `*` mit einem sinngerechten Verhalten für zwei Instanzen der Klasse (sinngerecht bedeutet, der `+` Operator soll z.B. nicht auf einmal Elemente dividieren). Füge den Funktionen auch hier Kommentare hinzu, die das Verhalten der Operatoren erklären, besonders auch, wenn die Instanzen zwei unterschiedlich große Arrays besitzen.

j) Erstelle weitere Operatorüberladungen zum Beispiel für die `+=`, `-=` und `*=` Operatoren sowie für Berechnungen zwischen der Klasse und einem einzelnen Double-Wert.