Classes, Objetos e Encapsulamento

1. Crie uma classe `Rectangle` com atributos `length` e `width`, cada um dos quais assume o padrão de 1. Forneça funções-membro que calculam os atributos `perimeter` e `area` do retângulo. Além disso, forneça as funções `set` e `get` para os atributos `length` e `width`. As funções `set` devem verificar se `length` e `width` são números de ponto flutuante maiores que $0,0$ e menores que $20,0$.

2. Implemente em C++ uma classe chamada `Aquecedor`. Ela deve ter um único atributo chamado `temperatura`, cujo tipo deve ser um ponto flutuante de precisão dupla. Defina um construtor que não recebe parâmetros e inicializa a temperatura em 20 graus. Crie os métodos aquecer e resfriar que aumentam e diminuem a temperatura em 5 graus, respectivamente. Defina um método para retornar o valor da temperatura.

3. Altere a classe do exercício anterior para que ela tenha três novos atributos: `temperatura mínima`, `temperatura máxima` e `fator de incremento da temperatura`. Os dois primeiros devem ser inicializados com 10 e 40 graus respectivamente no construtor. A classe deve ter um construtor sem parâmetros, que definirá o fator de incremento em 5 graus, um segundo construtor que recebe a temperatura inicial e um terceiro que recebe a temperatura inicial e o fator de incremento.

   Altere os métodos existentes na classe de forma apropriada com o objetivo de manter o estado do objeto sempre válido (ex: o fator de incremento deve ser usado toda vez que os métodos aquecer e resfriar forem chamados). Escreva mensagens na saída padrão quando uma ação não puder ser executada por não ser um estado de objeto válido.

   Por fim, crie um método que permita alterar o fator de incremento da temperatura depois de um objeto já ter sido criado.

4. Crie uma classe `SavingsAccount`. Utilize um membro de dados `static annualInterestRate` para armazenar a taxa de juros anual para cada um dos correntistas. Cada membro da classe contém um membro de dados `private savingsBalance` para indicar a quantia que os correntistas têm atualmente em depósito. Forneça a função-membro `calculateMonthlyInterest` que calcula os juros mensais multiplicando o `savingsBalance` pelo `annualInterestRate` dividido por 12; esses juros devem ser adicionados a `savingsBalance`. Forneça uma função-membro `static modifyInterestRate` que configura o `static annualInterestRate` com um novo valor. Escreva um programa `main` para testar a classe `SavingsAccount`. Instancie dois objetos diferentes da classe `SavingsAccount`, `saver1` e `saver2`, com saldos de $\$\ 2.000,00$ e $\$\ 3.000,00$, respectivamente. Configure o `annualInterestRate` como 3%. Em seguida, calcule os juros mensais e imprima os novos saldos de cada um dos correntistas. Então configure novamente o `annualInterestRate` como 4%, calcule os juros do próximo mês e imprima os novos saldos para cada um dos poupadores.

   Você pode utilizar o código abaixo (`main.cpp`) para testar o seu código.

```
1
2  #include <iostream>
3  #include "savings.h"
4
5  int main(void) {
6      // Main. Criando duas contas.
7      SavingsAccount saver1 = SavingsAccount(2000);
8      SavingsAccount saver2 = SavingsAccount(3000);
9
10     // Imprimindo o monthly balance
11     std::cout << saver1.calculateMonthlyInterest() << std::endl;
```

```
12      std::cout << saver2.calculateMonthlyInterest() << std::endl;
13
14      // Alterando atributo static publico
15      SavingsAccount::annualInterestRate = 3.00;
16
17      // Imprimindo o monthly balance. Tem que mudar para as duas classes.
18      std::cout << saver1.calculateMonthlyInterest() << std::endl;
19      std::cout << saver2.calculateMonthlyInterest() << std::endl;
20
21      return 0;
22  }
```

**Considerações!**

- Todos os exercícios devem conter `.h`, `.cpp`, e `main.cpp`;
- O seu `main.cpp` deve conter, minimamente, instruções para criação (instanciação de objetos) e chamadas das funções implementadas.