

Herança e Composição
----------------------

1. Analise o código disponibilizado nos slides da aula (hierarquia de classes: `Pessoa` e `Estudante`), entenda-o, e depois execute-o para ver/analisar os resultados. Em seguida:
  - Crie a Classe `Turma`, com atributos privados código (`String`), e ano (`int`). Crie um construtor que receba parâmetros para inicializar os atributos e os métodos de acesso e métodos modificadores (`gets` e `sets`).
  - Altere a classe `Estudante` para que tenha também um atributo privado `turma` do tipo `Turma`. Altere o construtor para receber um parâmetro que inicialize o novo atributo e crie os métodos de acesso e modificador para este novo atributo.
  - Altere o `main` para tratar esse novo atributo da classe `Estudante`.

```
1 #include <iostream>
2
3 #include "estudante.h"
4 #include "pessoa.h"
5
6 void f(Pessoa &pessoa) {
7     std::cout << "Na funcao: " << pessoa.defina_meu_tipo() << std::endl;
8 }
9
10 int main() {
11     Pessoa pessoa("Julio Reis.");
12     Estudante estudante("Jane Doe", 20180101);
13     std::cout << "A pessoa eh: " << pessoa.defina_meu_tipo() << std::endl;
14     std::cout << "O estudante eh: " << estudante.defina_meu_tipo() << std::endl;
15     f(pessoa);
16     f(estudante);
17
18     return 0;
19 }
```

2. Os serviços de correio expresso, como FedEx, DHL e UPS, oferecem várias opções de entrega, cada qual com custos específicos. Crie uma hierarquia de herança para representar vários tipos de pacotes. Utilize `Package` como a classe básica da hierarquia, então inclua as classes `TwoDayPackage` e `OvernightPackage` que derivam de `Package`. A classe básica `Package` deve incluir membros de dados que representam nome, endereço. Para simplificar nosso código, represente o endereço como uma única string. Além dos membros citados anteriormente, armazene dados que representam o peso (em quilos) e o custo por quilo para a entrega do pacote. O construtor `Package` deve inicializar esses membros de dados, em outras palavras, todos são argumentos do construtor. Assegure que o peso e o custo por quilo contenham valores positivos (faça uso de `unsigned int`). `Package` deve fornecer um método `public calculateCost` que retorna um `double` indicando o custo associado com a entrega do pacote. A função `calculateCost` de `Package` deve determinar o custo multiplicando o peso pelo custo (em quilos). A classe derivada `TwoDayPackage` deve herdar a funcionalidade da classe básica `Package`, mas também incluir um membro de dados que representa uma taxa fixa que a empresa de entrega cobra pelo serviço de entrega de dois dias. O construtor `TwoDayPackage` deve receber um valor para inicializar esse membro de dados. `TwoDayPackage` deve redefinir a função-membro `calculateCost` para que ela calcule o custo de entrega adicionando a taxa fixa ao custo baseado em peso calculado pela função `calculateCost` da super classe `Package`. A classe `OvernightPackage` deve herdar diretamente da classe `Package` e conter um membro de dados adicional para representar uma taxa adicional por quilo cobrado pelo serviço de entrega noturno. `OvernightPackage` deve redefinir a função-membro

`calculateCost` para que ela acrescente a taxa adicional por quilo ao custo-padrão por quilo antes de calcular o custo da entrega.

```
1 #include <iostream>
2 #include "Package.h"
3 #include "TwoDayPackage.h"
4 #include "OvernightPackage.h"
5
6 int main(){
7     Package package("Pacote 1", "Rua Passos, 71", 20, 15);
8     TwoDayPackage two_day_package("Pacote 2", "Av. PH Holfs, s/n", 5, 15, 10);
9     OvernightPackage over_night_package("Pacote 3", "Av. Santa Rita, 110", 50,
10         15, 15);
11
12     std::cout << package.calculateCost() << std::endl;
13     std::cout << two_day_package.calculateCost() << std::endl;
14     std::cout << over_night_package.calculateCost() << std::endl;
15
16     return 0;
17 }
```

3. Existe um sistema que realiza o controle de uma frota de aviões. Estes aviões, divididos em caças (aviões de combate) e jatos (aviões comerciais), possuem diversos atributos, tais como tamanho, velocidade máxima, capacidade, etc. Este fato é representado por uma hierarquia, a qual tem a classe **Aeroplane** como superclasse; e as classes **JetPlane** e **FighterAircraft** como subclasses.

O sistema funcionava bem até certo momento. No entanto, um estagiário, ao saber de sua iminente demissão, resolveu inserir bugs propositalmente, os quais vão desde memory leaks e erros lógicos, até erros que inviabilizam a compilação e funcionamento do sistema, tais como falhas de segmentação, violação de visibilidade, erros de sobrescrita, etc. Portanto, cabe a você corrigi-los para que se tenha novamente o funcionamento correto da frota. Ao fim, o sistema deve imprimir com sucesso características dos aviões cadastrados (elementos não inicializados corretamente podem ser descartados).

Os diferentes arquivos que compõem o sistema podem ser vistos da seguinte forma:

**main.cpp** - apenas um arquivo para inicializar o sistema; **fleet.h/fleet.cpp** - contém a classe **Fleet**. Ela representa uma frota de aviões, tendo métodos para composição da frota e exibição de dados dos aviões; **aeroplane.h/aeroplane.cpp** - contém a classe **Aeroplane**. Esta serve como base para classes que especializam os aviões; **fighteraircraft.h/fighteraircraft.cpp** - contém a classe **FighterAircraft**. Esta representa os aviões caça, sendo uma especialização da classe **Aeroplane**. Possui métodos e atributos específicos destes aviões **jetplane.h/jetplane.cpp** - contém a classe **JetPlane**. Esta representa os aviões comerciais a jato, sendo uma especialização da classe **Aeroplane**. Possui métodos e atributos específicos destes aviões.

4. Dado um simulador ultra-simplificado de uma fazenda com somente dois tipos de animais: vacas e cachorros, você deve consertar eventuais erros presentes no código. Nele possuímos quatro módulos:

**Animal**: super-classe da qual todas as outras classes de animais herdam. Cada animal possui um id único (`_id`) e uma cor (`_cor`). A variável `next_id` deve ser compartilhada entre TODAS as instâncias de **Animal**, ela é usada para definir o próximo id no método `get_new_id()` (que, novamente, deve ser compartilhada e usada por todas instâncias). Os métodos: `reproduz`, `faz_barulho` e `get_id` devem ser definidos pelas classes que herdam de **Animal** (somente estes).

**Vaca**: herda de **Animal**, possui um atributo extra `producao_leite`.

**Cachorro**: herda de **Animal**, deve definir somente os métodos virtuais da superclasse.

No arquivo **Animal.cpp**, existe uma função `popula()` que recebe um vector de ponteiros para **Animal\*** e o `popula`. Ele faz isso através da função `reproduz`, escolhendo um animal da fazenda aleatório para se reproduzir, essa operação deve ser repetida até que a população desejada seja atingida (`max_populacao`).

Adicione (ou remova) novos métodos se achar necessário.

## Considerações Gerais!

- Exercício individual.
- Entrega: conforme agendado no PVANET Moodle;
- Conforme estrutura abaixo apresentada crie um projeto para resolução de cada exercício (ex.: `pratica5_exercicio1.zip`, `pratica5_exercicio2.zip`, etc). Cada projeto deve conter os arquivos `.h`, `.cpp`, e `main.cpp` criados para resolução do exercício. Envie, através do PVANet Moodle, uma pasta compactada (`.rar` ou `.zip`) contendo todos os projetos (também compactados). A pasta compactada deve conter informações do aluno (ex.: `julio_reis-pratica5.zip`).

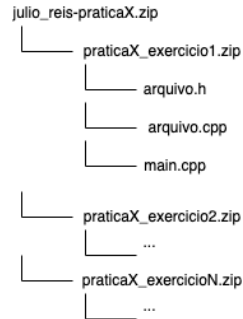


Figura 1: Estrutura de diretórios.

- O seu `main.cpp` deve conter, minimamente, instruções para criação (instanciação de objetos) e chamadas das funções implementadas (TODAS!!!). Para teste, você pode usar os exemplos fornecidos.