Depuração de Código: Herança, Composição, Polimorfismo

1. 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.

2. 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;
- Você deve entregar os exercícios separados por projeto, conforme exemplo abaixo (ex.: pratica7_exercicio1.zip, pratica7_exercicio2.zip, etc). Cada projeto deve conter todos os arquivos `.h`, `.cpp`, e `main.cpp` fornecidos. 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-pratica7.zip);

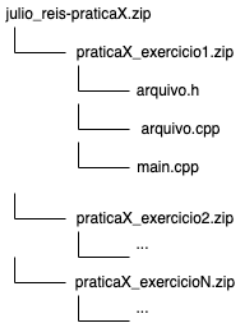

Figura 1: Estrutura de diretórios.

- O seu código não "corrigido"não deve conter erros de execução ou *leaks* de memória, etc.