UNIVERSIDADE FEDERAL DE SANTA CATARINA
CTS – DEC – ARA
ENC – Engenharia de Computação
Disciplina : Programação II
Professor : Antonio Carlos Sobieranski

Avaliação P1 – Programação Orientada à Objetos **Aluno: _____**

Avaliação Prova Teórica (PT) – Prazo para Entrega de ambas (PT e PP) as avaliações: 36 horas

Avaliação a ser desenvolvida individualmente, respostas em um PDF avulso com respostas enumeráveis. Respostas ilegíveis ou não passíveis de interpretação serão desconsideradas na correção.

---

1. **(1.0 ponto)** No que tange o Paradigma Orientado à Objetos (POO). Diferencie Classes, Instâncias e Objetos. *(min 4 linhas)*

---

2. **(1.5 pontos)** Defina o conceito de encapsulamento na POO ? Exemplifique fornecendo um exemplo código em C++ de sua autoria, no PDF a ser entregue, deixando claro a ocorrência deste.

---

3. **(1.5 pontos)** Em relação à Herança. Explique o conceito e apresente as suas principais suas vantagens, detalhando-as. *(min 5 linhas)*

---

4. **(1.5 pontos)** Quanto a questão anterior: Apresente um exemplo de 3 níveis com menos 2 construtores para cada classe (sobrecarga, desconsiderando o *default*). No main, instancie 5 objetos de tipos variados, realizando a chamada do construtor. *(escreva em C++, somente código)*

---

5. **(2.0 pontos)** O que é Polimorfismo e qual a sua relação com a Herança ? *(min 5 linhas)* Apresente 01 exemplo.

---

6. **(1.0 pontos)** Na Herança, a ordem de chamada dos construtores e destrutores é distinta. Justifique o porquê, apresentando também qual ordem as chamadas ocorrem.

---

7. **(1.5 pontos)** Verifique as afirmações abaixo em relação Programação Orientada a Objetos, colocando **V** ou **F**:
   - 1) ( ) A programação orientada a objetos tem como principais objetivos reduzir a complexidade no desenvolvimento de software e aumentar sua produtividade.
   - 2) ( ) Em C++, classes podem possuir quantos métodos construtores e destrutores forem necessários.
   - 3) ( ) Variáveis membro de uma classe com o qualificador *static* são únicas e reservadas nos objetos, não sendo possíveis serem acessadas externamente, mesmo quando qualificadas como públicas.
   - 4) ( ) Por meio do mecanismo de sobrecarga, vários métodos de uma mesma classe podem ter o mesmo nome, desde que suas listas de parâmetros sejam diferentes. No entanto, devem obrigatoriamente apresentar o mesmo tipo de retorno.
   - 5) ( ) Na herança, as subclasses tornam-se mais específicas em relação à superclasse, podendo adicionar novos atributos e métodos que serão acessíveis ao longo de toda a estrutura hierárquica.
   - 6) ( ) No uso de herança entre uma classe base e uma derivada, tanto construtores como destrutores são primeiro executados na classe base e depois na derivada.
   - 7) ( ) Em uma Herança de 3 níveis hierárquicos: é possível possuir várias instâncias de objetos em qualquer nível. Uma forma de polimorfismo é utilizar um ponteiro do tipo do primeiro nível hierárquico, e utilizá-lo para representar qualquer instância em qualquer nível hierárquico das subclasses, exceto uma instância do primeiro nível.
   - 8) ( ) No paradigma orientado à objetos a comunicação entre objetos se dá através de troca de mensagens.
   - 9) ( ) Herança múltipla é descrita na orientação à objetos como um mecanismo onde uma classe derivada herda a partir de duas classes bases, e possui características de ambas. Porém, nenhuma Linguagem de Programação implementa esse mecanismo devido ao auto nível de ambiguidade.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CTS – DEC – ARA
ENC – Engenharia de Computação
Disciplina : Programação II
Professor : Antonio Carlos Sobieranski

Avaliação P2 – Programação Orientada à Objetos **Aluno:** _____

Avaliação Prova Prática (PP) – Prazo para Entrega de ambas (PT e PP) as avaliações: 36 horas

Enunciado: Um novo sistema computacional para agendamento de tarefas de um super-computador foi idealizado para receber tarefas em lote e processá-las. As tarefas vem uma seguida de outra, e o computador vai processando elas conforme os recursos estiverem disponíveis para tal. Pode estar livre para processar imediatamente, ou estar tão sobrecarregado, que o processamento destas ocorre conforme for possível.

Basicamente, existem 2 políticas para definir como o sistema computacional executa suas requisições:

Política 1: primeira tarefa a chegar na lista será a **primeira** a ser executada, segunda tarefa a chegar será **segunda** a ser executada, e assim sucessivamente. *"Essa é a política da ordem, como em uma fila de banco"*.
Política 2: última tarefa a chegar na lista será a **primeira** a ser executada, penúltima a **segunda**, antepenúltima a **terceira**, e assim sucessivamente. *"Essa é a política da urgência, tudo pra ontem"*.

Desta forma, o sistema conta com o seguinte MENU que deverá ser implementado:

TASK SCHEDULER SYSTEM – UFxC TSS

Select an option below:
1. Schedule a task to process
2. Pick up a task to process using Policy 1
3. Pick up a task to process using Policy 2
4. Print pending tasks
5. Exit

onde:

- Opção 1 – insere o ID (ID: número identificador) de uma tarefa em uma lista única (std::vector) do tipo *size_t (*inserir um inteiro positivo somente*).* **Não pode ser inserido na lista um ID com número já existente. Se for o caso deve aparecer mensagem de erro e não inserir.**
- Opção 2 – selecionar no vector um ID de processo de acordo com a política 1 para ser processado, e remover ele da lista.
- Opção 3 – selecionar no vector um ID de processo de acordo com a política 2 para ser processado, e remover ele da lista.
- Opção 4 – mostrar em tela o vetor com os ID's restantes a serem processados, precedidos pelo ID do vector em ordem de posição (original do vector). Ou seja:
  Vector Position 0 – Job 5
  Vector Position 1 – Job 4
  Vector Position 2 – Job 9
  Vector Position 3 – Job 3

Requisitos da implementação:

- Implementar em Linguagem C++ e utilizando Orientação à Objetos
- Muito desejável usar herança, sob penalização de nota.
- Bastante desejável usar e demonstrar polimorfismo, sob penalização de nota.

- Utilizar uma única estrutura de dados vector para armazenar os **jobs IDs**.

- SUPER DICA de Herança e Polimorfismo: implementar uma única classe base, que acomode a estrutura de dados, chamado de **Scheduler**. Implementar 2 classes derivadas **Policy1** e **Policy2**, para implementar o comportamento, ou seja, as **políticas 1 e 2**.

Avaliação a ser desenvolvida individualmente. Submeter via moodle o código fonte da solução do problema. Código deve compilar corretamente para poder ser corrigido.

Preparar um breve vídeo de demonstração, mostrando rapidamente o código (bem superficialmente) seguido da execução do mesmo.

Código-fonte ser'compilado com: **g++ *.cpp -o exe**
Enviar código fonte somente, compactado em um único zip.
Será testado o programa através do menu requerido acima.
Programa não compilou é zero.
Dica: assistir a aula de polimorfismo.