Laboratório de Programação Orientada por Objetos (MIEIC)

7 de Junho de 2013

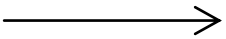
Mini-teste de escolha múltipla - Sem consulta - 60 Minutos

Assinala com um "X" a resposta correta a cada uma das perguntas seguintes. Cada pergunta só tem uma resposta correta. Respostas certas valem 1 ponto. Respostas erradas descontam 0.25 pontos. Para apagar uma resposta anteriormente introduzida, selecione a opção "Não quero responder a esta questão".

Nome: _____

Diagramas de Classes UML

1. Nos diagramas de classes UML, o seguinte símbolo representa que tipo de relação entre classes?

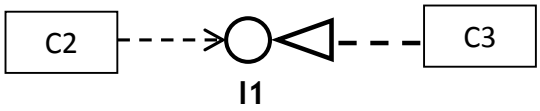


	a. associação navegável
	b. concretização (*realization)*
	c. generalização
	d. dependência

2. Nos diagramas de classes UML, a relação de generalização permite relacionar que tipos de elementos?

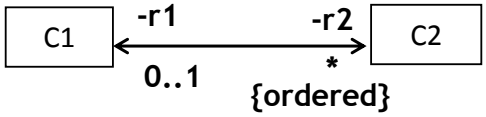
	a. subclasses e superclasses
	b. classes e interfaces
	c. objetos e classes
	d. instâncias de classes genéricas e classes genéricas

3. Qual é o significado do seguinte diagrama de classes UML:



	a. a classe C2 usa uma interface I1 que é implementada pela classe C3
	b. a classe C2 implementa a interface I1 e a classe C3 tem uma associação com I1
	c. a classe C2 depende da interface I1 e a classe C3 tem uma associação com I1
	d. a classe C2 interage com a classe C3 pela porta I1

4. Qual a tradução mais correta em Java do seguinte diagrama de classes UML:



	a. class C1 { private List<C2> r2; } class C2 { private C1 r1; }
	b. class C1 { private Set<C2> r2; } class C2 { private C1 r1; }
	c. class C1 { private C2 r2; } class C2 { private List<C1> r1; }
	d. class C1 { private Queue<C2> r2; } class C2 { private C1 r1; }

Diagramas de Sequência UML

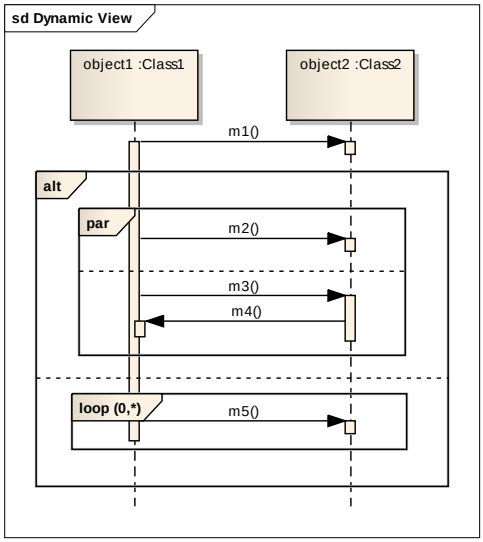
5. Os diagramas de sequência UML são mais apropriados para:

	a. mostrar sequências de instruções
	b. mostrar os estados por que os objetos podem passar ao longo da sua vida
	c. mostrar padrões de trocas de mensagens entre objetos num determinado contexto
	d. mostrar uma configuração de objetos e ligações entre objetos

6. Qual dos seguintes não é um tipo de mensagem válida em diagramas de sequência UML?

	a. mensagem de chamada de operação
	b. Mensagem de retorno de operação
	c. Mensagem de envio de sinal
	d. Mensagem de criação de classe

7. Qual das seguintes sequências de mensagens não é permitida pelo seguinte diagrama de sequência?



	a. m1
	b. m1 m3 m2 m4
	c. m1 m5 m5 m5
	d. m1 m4 m3 m2

8. Nos diagramas de sequência UML, o seguinte símbolo representa que tipo de mensagem?

	a. mensagem síncrona
	b. mensagem assíncrona
	c. mensagem de criação de objeto
	d. mensagem de fluxo de dados

Diagramas de Estados UML

9. Os diagramas de estados UML são mais apropriados para

	a. modelar o ciclo de vida de objetos ou sistemas
	b. modelar a interação entre objetos
	c. modelar algoritmos
	d. enumerar as variáveis de estado de um objeto

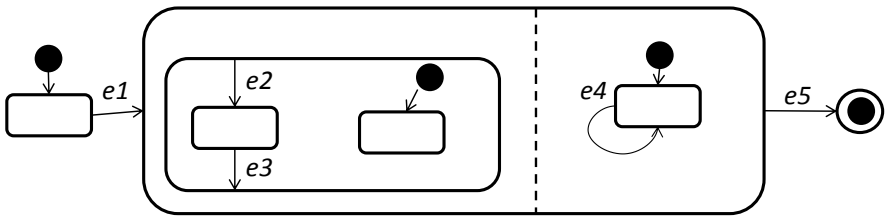
10. Nos diagramas de estados UML, a utilização de regiões ortogonais é útil para:

	a. evitar a explosão combinatória de estados
	b. evitar a explosão combinatória de transições
	c. evitar a explosão combinatória de eventos
	d. modelar eventos temporais absolutos e relativos

11. Nos diagramas de estados UML, qual dos seguintes elementos não pode ser usado numa transição?

	a. ação
	b. condição de guarda
	c. atividade
	d. evento temporal

12. Quais das seguintes sequências de eventos <u>não é</u> aceite pelo seguinte diagrama de estados?



	a. e1 e4 e3 e5
	b. e1 e2 e5
	c. e1 e2 e3 e4 e5
	d. e1 e4 e2 e2 e4 e3 e5

Padrões de Desenho

13. Um padrão de desenho surge com o propósito de:

	a. Reutilizar software para poupar tempo de desenvolvimento.
	b. Resolver problemas recorrentes de uma forma universal.
	c. Sugerir soluções garantidamente boas para problemas recorrentes.
	d. Ajudar no refactoring de código legado.

14. Os padrões de desenho assentam nas "pedras basilares" da orientação por objetos. Estas são:

	a. Abstração, Polimorfismo, Herança e Encapsulamento
	b. Abstração, Herança, Reencaminhamento e Polimorfismo
	c. Abstração, Polimorfismo, Invocação Remota e Reencaminhamento
	d. Abstração, Herança, Invocação Remota e Polimorfismo

15. A melhor estratégia para usar padrões de desenho é:

	a. Usá-los sempre que possível, pois serão sempre uma vantagem.
	b. Tê-los presentes para os usar sempre que a necessidade surja.
	c. Usar só se for preciso fazer refactoring.
	d. Usar apenas se já tivermos tentado tudo para resolver o problema em mãos.

16. Quando usamos o SWING em Java para construir interfaces gráficas, quais os dois padrões de desenho mais utilizados?

	a. COMPOSITE e OBSERVER
	b. STRATEGY e VISITOR
	c. VISITOR e COMPOSITE
	d. OBSERVER e STRATEGY

Refactoring e *Code Smells*

17. Em que consiste *refactoring*?

	a. Efetuar alterações à estrutura interna de um programa para o tornar mais fácil de compreender e modificar, sem alterar o comportamento observável do programa.
	b. Efetuar alterações à estrutura interna de um programa para melhorar o comportamento observável do programa.
	c. Efetuar alterações à estrutura interna de um programa para garantir que passa nos testes.
	d. Efetuar alterações à formatação de um programa para o tornar mais fácil de compreender e modificar, sem alterar o comportamento observável do programa.

18. Qual dos seguintes *code smells* não pode ser resolvido pelo *refactoring Extract Method*?

	a. Long Method
	b. Switch Statements
	c. Duplicated Code
	d. Lazy Class

19. Que *refactoring* considera ser o mais útil aplicar para melhorar o seguinte código?

```
if(hX==dX) {
      if(Math.abs(dY-hY) == 1){
            if(h.armado) {
                  d.vivo= false;
                  labi[dX][dY]= ' ';
            }
            else if(d.dorme == false)
                  h.vivo=false;
      }
}
....
if(hY==dY) {
      if(Math.abs(dX-hX) == 1) {
            if(h.armado) {
                  d.vivo= false;
                  labi[dX][dY]= ' ';
            }
            else if(d.dorme == false)
                  h.vivo=false;
      }
}
```

	a. Extract Method
	b. Pull-Up Method
	c. Replace Conditional with Polymorphism
	d. Undo Copy/Paste

20. Que *code smell* existe no seguinte código?

```
public class Customer {
   private Phone mobilePhone;

   public String getMobilePhoneNumber() {
      return "(" + mobilePhone.getAreaCode() + ") " +
             mobilePhone.getPrefix() + "-" +
             mobilePhone.getNumber();
   }  ...
}
```

	a. *Feature Envy*
	b. *Lazy Class*
	c. *Refused Bequest*
	d. *Duplicated Code*

JPF/AMA/JGB/NHF