Instituto Superior Técnico

Programação Avançada – 2008/2009

Primeiro Exame – 27/6/2009

Número: _____

Nome: _____

Escreva o seu número em todas as folhas da prova. O tamanho das respostas deve ser limitado ao espaço fornecido para cada pergunta. Pode usar os versos das folhas para rascunho. A prova tem **6** páginas e a duração é de **2.0 horas**. A cotação de cada questão encontra-se indicada entre parêntesis. Boa sorte.

1. (2.0) Considere os conceitos de **Sistema Computacional** e **Meta-Sistema Computacional**.

   (a) (1.0) Defina **Sistema Computacional**. Qual é a relação entre um programa e um sistema computacional?

   Um Sistema Computacional é um sistema que actua num determinado domínio.

   Um programa não é um sistema computacional mas sim uma descrição de parte de um sistema computacional. A execução de um programa já constitui um sistema computacional.

   (b) (1.0) Defina **Meta-Sistema Computacional**. Dê **três** exemplos de meta-sistemas computacionais.

   Um Meta-Sistema Computacional é um sistema computacional cujo domínio é outro sistema computacional.

   Como exemplos, podemos referir um* debugger*, um* profiler *e um* inspector*.

2. (2.0) Distinga **introspecção** de **intercessão** e dê um exemplo prático de cada.

   Introspecção é a capacidade de um programa de examinar a sua própria estrutura e comportamento. Como exemplo, podemos considerar um mecanismo para determinar a lista de parâmetros de uma função.

   Intercessão é a capacidade de um programa de modificar a sua própria estrutura e comportamento. Como exemplo, podemos considerar um mecanismo para mudar a classe a que uma instância pertence.

3. (1.0) A **Auto-Modificação** (*self-modification*) é uma capacidade extremamente poderosa disponibilizada por algumas linguagens de programação. No entanto, essa capacidade também acarreta alguns problemas. Discuta dois desses problemas.

   Um dos problemas é que a compilação fica muito dificultada pois o programa não tem uma forma estável.

   Outro dos problema é que a depuração fica também muito dificultada pelo facto de que o programa onde o erro foi despoletado pode já estar muito diferente daquele que foi escrito pelo programador.

4. (3.0) Pretende-se implementar um mecanismo de interligação com Java que permita a um programa escrito noutra linguagem criar objectos Java, obter referências para esse objectos no seu próprio espaço de endereçamento e usar essas referências para fazer invocações de métodos Java sobre os objectos correspondentes, recebendo os resultados dessas invocações (que poderão ser referências para outros objectos Java).

   Descreva uma possível implementação deste mecanismo de interligação. Não é necessário apresentar exemplos de código mas inclua toda a informação que considerar relevante para que outra pessoa possa realizar essa implementação sem problemas.

5. (3.0) Considere a implementação de um mecanismo de *undo* computacional para programas Java. Este mecanismo permite, em qualquer momento da execução de um programa Java, desfazer um dado número de modificações que tenham sido realizadas nos *fields* dos objectos criados pelo programa.

   Este mecanismo é operado a partir de um método estático que tem como parâmetro o número (inteiro) de modificações que se pretendem desfazer e estas são desfeitas por ordem cronológica inversa, i.e., a última modificação que tenha sido feita é a primeira a ser desfeita.

   Proponha um mecanismo de intercessão baseado em Javassist que implemente este mecanismo de *undo*. Não é necessário descrever exaustivamente a implementação do seu mecanismo mas inclua toda a informação que considerar relevante para que outra pessoa (conhecedora de Javassist) possa realizar essa implementação sem problemas.

6. (1.0) Distinga os seguintes conceitos da linguagem CLOS: função genérica, método, método primário, método auxiliar, método aplicável e método efectivo.

   Uma função genérica é uma função cujo comportamento depende dos tipos dos argumentos. A função genérica é implementada por um conjunto de métodos que correspondem a especializações da função genérica para diferentes tipos de argumentos e para diferentes papéis. Estes papéis permitem classificar os métodos em métodos primários (que produzem o valor a retornar) e métodos auxiliares (que modificam ou controlam o comportamento dos primários).

   Quando se aplica uma função genérica, os argumentos são usados para seleccionar o subconjunto de métodos da função cujos especializadores emparelham com os argumentos. Estes métodos aplicáveis são são ordenados e combinados tendo em conta a sua especificidade e o seu papel (como primários ou auxiliares) para produzir o método efectivo que vai ser de facto aplicado aos argumentos.

7. (2.0) O que é um *cross-cutting concern*? Quais são os problemas causados pelos *cross-cutting concerns*? Explique.

   Um* cross-cutting concern *é um requisito de uma aplicação cuja implementação atravessa todo o sistema, afectando um largo conjunto de classes, objectos ou funções.

   Os problemas dos* cross-cutting concerns *são:

   - **Code Tangling *Um módulo implementa um* concern *fundamental e parte de outro que lhe é transversal.***
   - **Code Scattering *Um* concern *transveral é implementado por fragmentos espalhados em várias partes do sistema.***

   Destes problemas, resulta:

   - ***Baixa qualidade:* tangling *de código facilita erros.***
   - ***Baixa rastreabilidade: onde está o código que implementa um* concern*? A que* concern *diz respeito este código?***
   - ***Difícil reutilização: cada módulo inclui fragmentos que não têm a ver com a sua função principal.***
   - ***Difícil evolução: de módulos que estão espalhados pelo sistema e de módulos que incluem fragmentos de outros módulos.***

8. (3.0) Considere o seguinte aspecto:

```
aspect ShapeListening {
    private Vector<Screen> Shape.listeners = new Vector<Screen>();

    public void Shape.addListener(Screen sc) {
        listeners.add(sc);
    }

    pointcut changeShape(Shape s):
      call(void Shape+.set*(..)) && target(s);

    after(Shape sh): changeShape(sh) {
        for(Screen sc : sh.listeners) {
            sc.redraw(sh);
        }
    }
}
```

(a) (2.0) A linguagem AspectJ é uma extensão da linguagem Java destinada à programação orientada a aspectos e introduz uma série de novos conceitos, em particular, *join point*, *pointcut*, *advice*, *inter-type declaration* e *aspect*. Explique estes conceitos e assinale-os (com círculos e setas) no exemplo acima.

(b) (1.0) Descreva, em linguagem corrente, os efeitos do aspecto apresentado.

9. (3.0) Um avaliador meta-circular permite, com bastante facilidade, experimentar diferentes semânticas para uma linguagem de programação.

   (a) (1.0) O que é um *avaliador meta-circular*? Explique.

   Um avaliador (ou interpretador) de uma linguagem é um procedimento que, quando aplicado a uma expressão dessa linguagem realiza as operações descritas nessa expressão.

   Um avaliador meta-circular é um avaliador definido na mesma linguagem que ele avalia.

   (b) (1.0) A presença de funções de ordem superior numa linguagem de âmbito dinâmico pode provocar dois problemas sérios, um quando uma função é passada como argumento e outro quando a função é devolvida como resultado. Que problemas são esses e em que condições ocorrem? Explique-os.

   O primeiro problema é o* downward funarg problem *e o segundo é o* upward funarg problem*.

   (c) (1.0) O que é uma *macro*? Para que serve?

   Uma* macro *é uma função que recebe formas sintáticas como argumento e computa uma forma sintática como resultado, forma essa que é avaliada no lugar da invocação da macro

   A* macro *serve para definir transformações sintáticas, por exemplo, para definir novas estruturas de controle à custa de outras já existentes.