Instituto Superior Técnico

Programação Avançada

First Exam – 7/6/2013

Number: ________________________

Name: ____

Write your number on every page. Your answers should not be longer than the available space. You can use the other side of the page for drafts. The exam has **6** pages and the duration is **2.0 hours**. The grade for each question is written in parenthesis. Good luck.

1. (2.0) Explain the concepts of *introspection* and *intercession* and provide one example for each of them.

   - ***A Introspecção é a capacidade de um sistema de* observar *a sua própria estrutura e comportamento. Por exemplo, saber "Quantos parâmetros tem a função* `foo`*?" é introspecção.***
   - ***A Intercessão é a capacidade de um sistema de* modificar *a sua própria estrutura e comportamento. Por exemplo, "Mudar a classe desta instância para* `Bar`*" é intercessão.***

2. (1.0) In order to allow introspection, the Java language had to reify the concept of class and had to provide three different ways for obtaining a class. Write fragments of Java programs that show each of these three different ways of obtaining a class.

```
"I am a string".getClass()

String.class

Class.forName("java.lang.String")
```

3. (1.0) In an object oriented language, some classes might represent objects with a lot of functionality but very little data, while other classes might represent objects containing a lot of data but little functionality. These different classes can be compared by computing, for a given class, the *functionality* ratio $\frac{m}{f+m}$, where m is the total number of public *methods* and f is the total number of public *fields*. A ratio of 1 means that the instances of the class do not contain public fields, while a ratio of 0 means that the instances do not contain public methods.

   Write, in Java, a class named `Metrics` containing a static method named `functionality`. This method accepts a `Class` as an argument and returns a double expressing the *functionality* ratio of the class.

```
import java.lang.reflect.*;

class Metrics {
    public static double functionality(Class c) {
        int m = c.getMethods().length;
        return ((double) m)/(m + c.getFields().length);
    }
}
```

4. (1.0) Consider the following fragment of Java code:

```
Class cls = Class.forName("FooBar");
Method meth = cls.getMethod("baz");
meth.invoke(cls.newInstance());
```

   Rewrite the fragment so that you obtain the same result but without using any reflection mechanisms.

```
new FooBar().baz();
```

5. (2.0) There are several proposals for the inclusion of *multiple dispatch* in Java. Explain this concept and write about the advantages and disadvantages of its implementation in Java. In particular compare the concept with the already existent concept of *overloading*.

Multiple dispatch vs single dispatch plus overloading.

Run time types vs compile time types.

Cascaded ifs plus instanceof vs double dispatch pattern vs reflection.

6. (2.0) Some programming languages allow an instance of a class to become a member of a different class. Explain the advantages and disadvantages of that possibility.

As vantagens incluem:

- ***Aumento de poder expressivo: permite escrever programas mais simples do que quando isso não é possível.***
- ***Possibilidade de criar instâncias quando ainda não é bem conhecida a classe a que devem pertencer e poder especificar essa classe mais tarde.***
- ***Permitir actualizar as instâncias já existentes sem ter de fazer* re-deploy *da aplicação***

As desvantagens incluem:

- ***Implementação mais complexa da linguagem.***
- ***Ineficiências no acesso às instâncias ou na mudança de classe.***
- ***Dificuldades de optimização.***
- ***Funcionamento do programa difícil de entender.***
- ***Depuração mais difícil.***

7. (1.0) CLOS provides the *protocol* `update-instance-for-redefined-class`. Why does it exist? What can we do with it? When is this protocol executed?

   The protocol exists because CLOS allows classes to be redefined at run time and instances need to know about that so that they can be updated. The protocol allows the programmer to write code that migrates the values of the previous slots of an instance to the slots prescribed by the redefined class. The protocol is not executed when the class is redefined. Instead, it is executed when an instance of a redefined class is accessed.

8. (2.0) In general, aspect-oriented programming allows not only the dynamic *cross-cutting* but also the static *cross-cutting*. Describe and explain the capabilities of each of these concepts.

   O* cross-cutting *dinâmico implica a modificação do comportamento de um programa, permitindo a inserção de comportamento adicional em pontos bem definidos da execução do programa.

   O* cross-cutting *estático implica a modificação da estrutura estática do programa, permitindo modificações na hierarquia de class, a adição de methods, a implementação de novas interfaces, etc.

9. (1.0) Describe, in your own words, the effect of the following aspect of AspectJ:

```
aspect FooAspect {
  static final int LIMIT = 100;

  before(Foo foo, int newBar):
      set(int Foo.bar) &&
      target(foo) && args(newBar) &&
      ! within(FooAspect) {
          if (newBar > LIMIT) {
              foo.bar = LIMIT;
          }
    }
}
```

10. (1.0) The Common Lisp language provides the forms `function` (`#'`) and `funcall`, while the Scheme language does not provide them. Why? Explain.

    Scheme has a single namespace for functions and variables, while Common Lisp has different namespaces for those. In Scheme, a name only means one thing at any given instant, but in Common Lisp it can mean several things and the language disambiguates by considering that a name in the first position of a non-atomic expression refers to a function while in the other positions it refers to a variable. The forms `function` ***and*** `funcall` ***are used to change this meaning.***

11. (2.0) Some languages, such as Pascal, allow functions to be defined inside other functions, but do not allow functions to be returned as values or to be stored in variables. Some other languages, such as C, allow functions to be returned as values or to be stored in variables, but do not allow functions to be defined inside other functions.

    What problem explains those two different approaches to language design? If you want to have a language that has the same capabilities of both Pascal and C, what should you do? Explain all implications.

    In order to be lexically scoped, the free variables of a function must be bound to the values that were available when the function was created. Given that it is impossible to predict the moment when the function is called, those values must be available until the function can no longer be called. Pascal saves those values on the stack and this means that functions cannot outlive the containing function, while C forces all free variables to have global scope, thus preventing internal function definitions.

    To solve the problem, each function must contain a reference to the lexical environment that was available when the function was created, and this environment must have indefinite extent, which usually requires garbage collection.

12. (1.0) Consider the `yield` operator as provided in Python. What does it do? What is it used for?

    The operator is used to allow a function to resume its computation from the point where it previously returned. Whenever a yield is reached, the function returns a value but also saves its execution state so that the next time it is invoked, it resumes from where it was.

    This is used to implement generators.

13. (1.0) Using the ambiguous operators `amb` and `fail`, define a function named `integer-between` that accepts two integers a and b as arguments and non-deterministically returns an integer i such that $a \leq i \leq b$.

```
(define (integer-between a b)
  (if (> a b)
    (fail)
    (amb a
         (integer-between (+ a 1) b))))
```

14. (1.0) What kind of problem is a good match for non-deterministic computation? Explain.

Problems where there are alternative ways for continuing a computation and where backtracking is required for findind the correct way that leads to a solution.

15. (1.0) **Julia** is a recently proposed programming language with metaprogramming capabilities. As an example, the following fragment defines three-argument versions of the two-argument predefined operators `foo` and `bar`:

```
for op = (:foo, :bar)
  eval(:($op(a,b,c) = $op($op(a,b),c)))
end
```

What must be the purpose of the symbols `:` and `$`? Explain.

Given that the example uses the `eval` ***function, its argument must be a fragment of code. In this case, the symbol*** `:` ***must have the same meaning as the*** **backquote** ***in Lisp, and*** `$` ***must have the same meaning as the*** **comma*****, i.e., the*** `:` ***does not evaluate the expression except in the subexpressions preceeded by*** `$`***. These symbols are being used for creating*** **templates** ***that simplify meta-programming.***