Exam 1

1. (1.0) What is the difference between a Computational System and a Computational Meta-System?

A computational system is a system that can be described by a set of rules that can be executed by a computer. A running program is a computational system.

A computational meta system is a computational system that has another system as target. It's a system that can reason about other systems. An example of a computational meta system is a debugger.

[**Professor answer:**]

A Computational System is a system that operates within a specific domain.
A program is not a computational system but rather a description of part of a computational system. The execution of a program already constitutes a computational system.
*A Computational Meta-System is a computational system whose domain is another computational system. Examples include a debugger, a profiler, and an inspector.

2. (1.0) In Java, several concepts, such as classes, methods, and fields, are reified. Why? Explain.

Reification means creating an instance of an entity of the object system. The objest system is the target system of a computational meta-system.
Classes, methods, and fields are reified in Java because they are objects in the object system. They are instances of the classes Class, Method, and Field, respectively.
Reification can be used to have information about the program at runtime. For example, you can get the class of an object at runtime using the method getClass().

3. (1.0) Java supports not only annotations but also meta-annotations. What is the purpose of meta- annotations? Explain and provide examples.

Meta-annotation in java are annotations that are used to annotate other annotations. They provide information about the annotation. For example, the annotation @Target is a meta-annotation that specifies the types of elements that an annotation can be applied to.

@Target: This meta annotation specifies the target elements to which an annotation can be applied. For example, @Target(ElementType.METHOD) restricts the annotation to be applied only to methods.

@Retention: This meta annotation specifies the retention policy for an annotation, determining how long the annotation should be retained. For example, @Retention(RetentionPolicy.RUNTIME) specifies that the annotation should be available at runtime.

4. (1.0) In Java it is possible to use introspection on the existing classes. What would happen if we could also use intercession? Explain the advantages and disadvantages

Introspection is the ability to analyze the structure of a class at runtime. It allows you to inspect the methods, fields, and annotations of a class.

Intercession, in the other hand, is not possible in Java because the bytecode is immutable. If intercession was possible, you could change the behavior of a class at runtime. Advantages of intercession are that you could add new methods to a class at runtime, or change the behavior of existing methods. Disadvantages are that it would make the code harder to understand and maintain, and it could lead to unexpected behavior.

5. (1.0) Julia is a recently proposed programming language. Julia's documentation include the following sentence:

> The key point here is that Julia code is internally represented as a data structure that is accessible from the language itself.
> **What capabilities can we expect from that feature? Explain.**

If the code is represented as a data structure that is accessible from the language itself, you can manipulate the code at runtime. You can generate code, modify code, or analyze code. This is useful for metaprogramming, where you write code that generates code. For example, you can write a macro that generates code based on the input arguments. In fact we wrote a metacircular evaluator that is an example of metaprogramming.

6. (2.0) The Common Lisp Object System implements the concepts of generic function, method, method combination, and effective method. Explain these concepts

CLOS or Common Lisp Object System is an object-oriented system for the programming language Common Lisp.

A generic function is a function that can have multiple methods. When calling a generic function, the actual method is called based on the arguments type. That's what we call multiple dispatch.
About the **method**, it's a function that is associated with a generic function. It's called when the generic function is called with the correct arguments.
Method combination in CLOS provides a mechanism to combine the results of multiple methods, offering flexibility in method invocation and result aggregation. It allows developers to customize

method invocation behavior, tailoring it to specific requirements or preferences.
Effective method is the method that is actually called when a generic function is invoked. It is the result of the method combination process, which determines the final method to be executed based on the applicable methods and their combination.

[Professor answer:]

A generic function is a function whose behavior depends on the types of the arguments. The generic function is implemented by a set of methods that correspond to specializations of the generic function for different types of arguments and different roles. These roles allow classifying methods into primary methods (which produce the return value) and auxiliary methods (which modify or control the behavior of primaries).

When applying a generic function, the arguments are used to select the subset of methods of the function whose specializers match the arguments. These applicable methods are sorted and combined based on their specificity and role (such as primaries or auxiliaries) to produce the effective method that will actually be applied to the arguments.

7. (1.0) Consider a programming language supporting metaclasses. Describe possible uses for methods defined at the metaclass level

[I don't think this question is relevant to the course, but I'll answer it anyway]
Meta-classes are classes of classes. They are used to define the behavior of classes. Methods defined at the metaclass level can be used to customize the behavior of classes. For example, you can define a method that is called when an instance of a class is created, or when a method is called on a class. This can be used to implement features like lazy evaluation, memoization, or logging.

[**Professor answer:**]

The meta-class of an object is the class of the class of that object. A meta-class is a class whose instances are classes.

The utility of metaclasses lies in the fact that they allow:

Determining the inheritance structure of the classes that are its instances.
Determining the representation of the instances of the classes that are its instances.
Determining access to the slots of the instances.

8. (1.0) In CLOS, generic functions can be specialized not only on the types of the arguments, but also on specific arguments. This is known as instance specialization. Describe a situation where instance specialization is useful.

Instance specialization allows you to specialize a method based on the value of an argument, not just its type. This can be useful when you want to provide different behavior for different instances of a class. For example, you could have a method that calculates the area of a shape, and specialize it for different shapes. You could have a different implementation for a circle, a square, or a triangle.

Exam 2

1. (3.0) Consider the following micro-evaluator:

```
(define (eval expr)
  (cond ((number? expr)
       expr)
      (else
       (apply
        (cond ((eq? (car expr) 'add) +)
            ((eq? (car expr) 'subtract) -)
            ((eq? (car expr) 'multiply) *)
            (else (error 'eval "Unknown operator")))
        (list-of-values (cdr expr))))))

(define (list-of-values exprs)
  (if (null? exprs)
    (list)
    (cons (eval (car exprs))
        (list-of-values (cdr exprs)))))
```

This micro-evaluator can evaluate simple arithmetic expressions such as:

> (eval '(add 1 (multiply 2 3) (subtract 5 2)))
> 10

(a) (1.0) How many calls to the function eval are needed to evaluate the previous expression?
8

Explanation:
The expression is (add 1 (multiply 2 3) (subtract 5 2)). The function eval is called for each sub-expression, and for the main expression. So, the function eval is called 4 times. The first call is for the main expression, and the other 3 calls are for the sub-expressions. Each sub-expression is evaluated by the function eval, and the result is passed to the main expression. So, the function eval is called 4

times in total. Then the function list-of-values is called 4 times, one for each sub-expression. So, the total number of calls to the function eval is 4 + 4 = 8.

(b) (1.0) Make the necessary changes to force the evaluator to use a right-to-left evaluation order for the arguments of the operators. This means that, in the previous expression, the sub-expression (subtract 5 2) must be evaluated before the sub-expression (multiply 2 3) which must be evaluated before the sub-expression 1.

```
(define (list-of-values exprs)
    (if (null? exprs)
            (list)
            (let ((rest-values (list-of-values (cdr exprs))))
                (cons (eval (car exprs))
                            rest-values))))
```

The change is in the function list-of-values. Instead of evaluating the first expression and then calling list-of-values recursively, we first call list-of-values recursively and then evaluate the first expression. This forces the evaluator to use a right-to-left evaluation order for the arguments of the operators.

(c) (1.0) Consider the implementation, in the previous evaluator, of a language extension identical to the arithmetic if aif. This is a control operator that exists in some popular programming languages (e.g., Fortran) that requires four expressions as arguments. The operators evaluates the first expression and, depending on its value, it evaluates just one the remaining expressions: if the value is negative, it evaluates the second expression, if the value is zero, it evaluates the third expression, otherwise, it evaluates the fourth expression.

> (eval ’(add 1
> (aif (subtract (multiply 2 3) (add 3 4))
> (add 4 5)
> 6
> (subtract 7 2))))
> 10

Redefine the evaluator to implement the operator aif.

```
(define (eval expr)
  (cond ((number? expr)
         expr)
        ((eq? (car expr) 'aif)
         (let ((r (eval (second expr))))
           (cond ((< r 0) (eval (third expr)))
                 ((= r 0) (eval (fourth expr)))
                 (else (eval (fifth expr))))))
        (else
         (apply
          (cond ((eq? (car expr) 'add) +)
                ((eq? (car expr) 'subtract) -)
                ((eq? (car expr) 'multiply) *)
                (else (error 'eval "Unknown operator")))
          (list-of-values (cdr expr))))))
(define (list-of-values exprs)
  (if (null? exprs)
      (list)
      (cons (eval (car exprs))
            (list-of-values (cdr exprs)))))
```

2. (1.0) The gensym function exists for a very long time in almost all Lisp dialects. What does it do? What is its purpose? Explain.

The gensym function is a functoin used to change the name of a symbol in order to make it unique. This is useful when the main goal is to avoid multiple evaluation. It's particulary used in the context of macros, where you want to generate new piece of code that doesn't interfere with the existing code.

[Prof answer:]
The gensym function returns a new unique symbol each time it is invoked. It is mainly used to define macros in order to avoid problems of multiple evaluation and undue capture of bindings.

3. (2.0) Several programming languages, such as, Haskell, O'Caml, F#, and Clojure, support lazy evaluation.

(a) (1.0) What is lazy evaluation? Explain

Lazy evaluation is a technique used in programming languages where expressions are not evaluated until their results are actually needed. This means that the evaluation of an expression is deferred until it is actually used in the program.

(b) (1.0) Explain a possible implementation for lazy evaluation in the metacircular evaluator presented in the course.

In a metacircular evaluator (an evaluator written in the same language it evaluates), when dealing with functions, it's important to handle lexical scoping correctly. This means that when a function is called, the arguments passed to it should be evaluated in the correct lexical environment (the environment where the function was defined, not where it's called).

To achieve this, the evaluator "wraps" each argument in a structure that associates the expression representing the argument with the environment in which it should be evaluated. This way, when the function is actually invoked, the "wrapped" arguments can be "unwrapped" by evaluating the associated expressions in their corresponding environments.

However, to avoid repeatedly evaluating the same expression, the "wrapper" should be able to store the result of the first evaluation, so that subsequent references to the same argument can simply return the stored result, instead of re-evaluating the expression.

4. (1.0) What is the meaning of the acronym REPL?

REPL means Read Eval Print Loop

- Read is the input phase, in which the program reads the input from the user.
- Eval is the evaluation phase, in which the program evaluates the input.
- Print is the output phase, in which the program prints the result of the evaluation.
- Loop is the loop phase, in which the program goes back to the read phase to read the next input.

5. (1.0) Aspect-oriented Programming was designed to deal with the problems caused by cross-cutting concerns, namely code tangling and code scattering. Explain these terms

- Code tangling: Code tangling refers to the situation where different concerns or aspects of a program are mixed together in the same code. This makes the code harder to understand, maintain, and modify. For example, if the logging code is mixed with the business logic code, it becomes harder to change the logging behavior without affecting the business logic.
- Code scattering: Code scattering refers to the situation where the code for a single concern or aspect is spread across multiple parts of the program. This makes it harder to locate and understand the code related to a specific concern. For example, if the error handling code is scattered across multiple functions, it becomes harder to understand and maintain the error handling behavior.

Exam 3

1. (1.0) In order to allow introspection, the Java language had to reify the concept of class and had to provide ways for obtaining a class, either from an instance or from the name of the class or from a String containing the name of the class. Write fragments of Java programs that show each of these three different ways of obtaining a class

```java
// Obtaining a class from an instance
Object obj = new Object();
Class<?> clazz1 = obj.getClass();

// Obtaining a class from the name of the class
Class<?> clazz2 = Object.class;

// Obtaining a class from a String containing the name of the class
String className = "java.lang.Object";
Class<?> clazz3 = Class.forName(className);
```

2. (1.0) Consider a hypothetical definition of a complexity metric for a class as the total number of public fields described in the class plus the total number of public methods described in that class. Write, in Java, a class named Metrics containing a static method named complexity. This method accepts a String with the name of a class and returns an integer with the complexity metric of that class

```java
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Person {
    public String name;
    public int age;

    public void sayHello() {
        System.out.println("Hello, my name is " + name);
    }

    public void sayAge() {
        System.out.println("I am " + age + " years old");
    }
}

public class Metrics {
    public static int complexity(String className) throws ClassNotFoundException {
        Class<?> clazz = Class.forName(className);
        Field[] fields = clazz.getFields();
        Method[] methods = clazz.getMethods();
        return fields.length + methods.length;
    }
}
```

3. (1.0) The Lisp language invented quote and, later, backquote. Which are their uses? How do they work?

- Quote: Is an operator that given an expression it doesn't evaluate it. It returns the expression as is. For example, if you have a list (1 2 3) and you quote it, it will return the list (1 2 3) instead of evaluating it.
- Backquote: Is an operator that allows you to evaluate an expression, but selectively quote parts of it. You can use the comma operator to evaluate an expression inside a backquoted expression. For example, if you have a variable x with the value 42, you can write `(list 1 2 ,x 3)` and it will return the list (1 2 42 3).

They are particularly useful when writing macros and performing metaprogramming.

4. (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 it's a feature of some programming languages where the method that has to be called is selected based on the type of the argument. For example, having a method that receives two arguments, the method that will be called is the one that matches the types of the two arguments. In Java, there is the concept of overloading, where you can have multiple methods with the same name but different arguments.
In Java, overloading a method is based on the static type of the arguments, not the runtime type. This means that the method to be called is selected at compile time, based on the static type of the arguments. In multiple dispatch, the method to be called is selected at runtime, based on the runtime type of the arguments. This allows for more flexibility and extensibility, as you can add new methods without modifying the existing ones. However, it can also make the code harder to understand and maintain, as the method to be called is not determined by the method signature, but by the runtime type of the arguments.

5. (2.0) In CLOS, the application of a generic function entails the computation of the effective method. Describe the necessary steps for that computation.

Effective method is the result of the method combination process, which determines the final method to be executed based on the applicable methods and their combination. The steps for computing the effective method in CLOS are as follows:

- a: The applicable methods are selected. An applicable method is a method whose specializers match the arguments of the generic function.
- b: The selected methods are arranged from most specific to least specific. Specificity is determined by the order of the specializers in the method definition.
- c: The arranged methods are combined, producing the effective method.
  For example, imagine a method called `calculate-area` that has two methods, one for calculating the area of a circle and another for calculating the area of a square. When `calculate-area` is called with a circle, the method for calculating the area of a circle is selected and executed. When `calculate-area` is called with a square, the method for calculating the area of a square is selected and executed. The effective method is the method that is actually executed based on the arguments passed to the generic function.

6. (3.0) We want to adapt the object-oriented model of the Java language to allow the use of active values. An active value is a Java object that, when modified, notifies other objects of that event. As an example, consider a Java object that represents a car and which class has the following definition:

```
class Car extends ... {
String brand;
String model;
...
float currentSpeed;
...
}
```

Given an instance of a car, we want to attach to it one or more objects (which we will call listeners) so that these objects will be notified via some pre-defined method call that will receive, as argument, the modified object. For example, it should be possible to attach to a specific car a graphical object that continuously represents the actual speed of that car. Given this scenario, suggest an intercession mechanism based in Javassist that allows the cre- ation of active values. It is not necessary to present the implementation of the mechanism but include all the necessary information that is relevant for someone else to implement the mecha- nism you propose.

Idea: You can use Javassist to create a proxy object that intercepts the setter methods of the Car class. When a setter method is called, the proxy object can notify the listeners that the object has been modified. The proxy object can keep a list of listeners and call a method on each listener when a setter method is called. The listeners can then update their representation of the car based on the new values.
Moreover you can use the Observer pattern to implement the listeners. The Observer pattern is a behavioral design pattern that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

9. (2.0) The majority of programming languages employ two concepts related to the binding of names and values: definition and assignment.

(a) (1.0) In terms of the programming language, what is the difference between these two concepts?

Definition: Definition refers to the act of introducing a new name into the program's scope and specifying the initial value it represents. When a variable is defined in a program, it allocates memory for the variable and associates it with a specific data type or value. This typically occurs at the point where the variable is first declared or initialized.

Assignment: Assignment, on the other hand, involves changing the value of an existing variable. It does not create a new variable but rather modifies the value stored in a variable that has already been

defined. Assignments are used to update the value of a variable as the program executes, allowing for dynamic changes to data stored in memory.

(b) (1.0) In terms of the implementation of the language in a meta-circular evaluator, what is the difference between these two concepts?

Definition: In the implementation of a meta-circular evaluator, definition involves introducing a new binding between a name and a value in the evaluator's environment or symbol table. This may include adding a new entry to the environment that associates a name with a value, allowing subsequent evaluations to access and manipulate that value.

In julia, for example, you can define a variable using the `let` keyword. This will create a new binding in the environment.

Assignment: Assignment, in the context of a meta-circular evaluator, typically involves updating the value associated with an existing name within the evaluator's environment. This update might occur as the result of evaluating an assignment statement in the language being evaluated. It modifies the existing bindings within the environment without creating new ones.

11. (1.0) Discuss the differences between external iterators and internal iterators

An Internal Iterator is a higher-order function that applies a function to each element of a collection.
– The iteration is controlled by the producer of values.
– Used in Lisp, Haskell and other Functional Languages.
In Javascript, map, filter, and reduce are examples of internal iterators.

An External Iterator is an accessor of the current element of a collection, and, potentially, of the next one.
– The iteration is controlled by the consumer of values.
– Used in C++, Java and other Object-Oriented Languages.
In Java, the Iterator interface is an example of an external iterator.

13. (2.0) Consider the ambiguous operator amb proposed by John McCarthy, that non-deterministically returns the value of one of the expressions provided as arguments.

(a) (1.0) Using the amb operator, define a function called integer-bigger-or-equal-to that accepts an integer as argument and non-deterministically returns an integer bigger or equal to its argument.

```
(define (integer-bigger-or-equal x)
  (amb x (+ x 1) (+ x 2) (+ x 3)))
```

or

```
(define (integer-bigger-or-equal-to x)
  (amb x (integer-bigger-or-equal-to (+ x 1))))
```

prof:

```
(define (inteiro-maior-ou-igual-a a)
    (amb a
    (inteiro-maior-ou-igual-a (+ a 1))))
```

(b)1.0 Consider the operator fail. Together with the operator amb, define a function list-element that receives a list as argument and non-deterministically returns one element of that list.

```
(define (list-element lst)
    (amb (car lst) (list-element (cdr lst))))
```

prof:

```
(define (elemento-lista l)
    (if (null? l)
        (fail)
        (amb (car l)
            (elemento-lista (cdr l)))))
```

So if have a null list, it will fail. Otherwise, it will return the first element of the list or call itself recursively with the rest of the list.

Exam 4

1. (1.0) What is reflection? What are the degrees of reflection you know? Explain and provide examples.

Reflection is a particular feature of programming language that makes possible to analyze the structure of a program during its execution, so at runtime. It allows to inspect and modify the structure of a program, such as classes, methods, fields, and annotations. Reflection is used in metaprogramming, debugging. There are 2 types of reflection: introspection and intercession.

- Introspection: it's the ability to inspect the structure of a program at runtime. For example, you can get the class of an object, the methods of a class, or the annotations of a method.
- Intercession: it's the ability to modify the structure of a program at runtime. For example, you can create new classes, add new methods to a class, or change the behavior of a method.

3 (2.0) The Java language allows method overloading, while the Common Lisp language permits multiple dispatch. What are the similarities and differences between these two approaches? How could you implement multiple dispatch in Java?

Both aim to do the same thing, which is to call a method based on the types of the argument. The way they do it is different.

- multiple dispatch: it calls the method based on the **run time** type of the arguments
- overloading: it calls the method based on the **compile time** type of the arguments

To implement multiple dispatch in Java, one could use cascades of instanceof with casts, or employ double dispatch patterns, or utilize reflection to dynamically identify the best method for a given set of arguments.

5. (2.0) Distinguish the following concepts in the CLOS language: generic function, method, primary method, auxiliary method, applicable method, and effective method.

A generic function is a function which behavior depends on the type of the arguments. The generic function is implemented using a combination of methods. The combination consist of primary methods, which are the methods that returns a value and applicable methods, that are methods that modify the behavior of the primary methods. The applicable methods are a subset of the methods that are applicable to the arguments of the generic function. The effective method is the method that is actually called when the generic function is invoked. It is the result of the method combination process, which determines the final method to be executed based on the applicable methods and their combination.

6. (1.0) What is a meta-class? What is the usefulness of meta-classes? Explain

A meta class is a class of a class. It's a class whose instances are classes. The utility lies in:

- determining the inheritane structure of the classes that are its instances. For example, you can determine if a class is a subclass of another class.

- determining the representation of the instances of the classes that are its instances. For example, you can determine the slots of a class.
- determining access to the slots of the instances. For example, you can get or set the value of a slot of an instance of a class.

1. (1.0) Which are the advantages and disadvantages of having a language that permits to change the class of an instnae at runtime? Explain

Pros:

- more expressive power: you can change the behavior of an object at runtime, which can be useful in certain situations.
- more flexibility: you can adapt the behavior of an object to different situations without having to create new classes.
- more dynamic: you can create new classes and change the behavior of objects at runtime, which can be useful in metaprogramming.

Cons:

- hard to maintain
- hard to debug
- hard to optimize
- hard to understand
- can lead to unexpected behavior

Exam 5

1. Differences betweeen comptuational system and computational meta-system

A computational system it's a system that acts and reasons in a particular domain.
A program is NOT a computational system but's it's a description of a computational system. Instead, a **running** program is considered a computational system.

A computational meta system it's a system which domain is another computational system, and for example we can have:

- debugger
- profiler
- inspector

2. Differences between introspection and intercession

- introspection: it's the ability of a program of analyzing its own structure at runtime
- intercession: it's the ability of a program to modify its own structure at runtime

3. (1.0) Self-modification is an extremely powerful capability provided by some programming languages. However, this capability also brings about some problems. Discuss two of these problems.

Self modification it's the ability of a program to modify its own structure at runtime. This can be useful in metaprogramming, where you write code that generates code. However, self-modification can also lead to some problems.
The main problems are that the compilation is kinda hard since the structure of the program is not stable. Also, the debugging is hard since the program can change its structure at runtime and when we are going to debug the program we are going to see the program changed, so good luck in finding where is the problem.

9. Metacircular evaluator

a. What is a metacircular evaluator?
An evaluator is a program that can, given an expression, it computes the result of that expression and returns the value. A metacircular evaluator is an evaluator that is written in the same language it evaluates. For example, a metacircular evaluator for Lisp would be written in Lisp.

(b) (1.0) The presence of higher-order functions in a dynamically scoped language can cause two serious problems, one when a function is passed as an argument and another when the function is returned as a result. What are these problems and under what conditions do they occur? Explain them.

Let's say that an **high order function** is a function that can receive another function as an argument or return a function as a result. The problems are:

- The first is the downward funarg problem, which occurs when a function is passed as an argument and then called in a different context, where the variables of the original context are not available. This can lead to unexpected behavior and errors.
- The second is the upward funarg problem, which occurs when a function is returned as a result and then called in a different context, where the variables of the original context are not available. This can also lead to unexpected behavior and errors.

c. what is a macro? what's it's purpose?
A macro is a function that given an expression as base it computes another expression starting from

the base one. It doesn't evaluate the base expression but it evaluates the resulting expression. Macro's job is to define transformations on the code. It's used in metaprogramming, where you write code that generates code.

Exam 6

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 is based on the runtime type, instead the single dispatch that is based on the compile time type. The main advantage of multiple dispatch is that it allows to select the method to be called based on the runtime type of the arguments. Method overloading, on the other hand, is based on the compile time type of the arguments.

Talking about the implementation:

- cascade of instance of: checking the type of the arguments and calling the correct method based on the instance of the arguments. But this is not a good solution since it's not scalable and adding a new type involves changing a lot of code.
- double dispatch: It's based on designing a hierarchy of clases and interfaces that allows to select the correct method based on the type of the argument. The invocation of the method is determined dispatching the method on both the receiver and the argument.
- Reflection: Using reflection to dynamically determine the method to be called based on the runtime type of the arguments. This is not efficient and can lead to performance issues.

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

In Scheme there is only a single namespace, while in Common Lisp there are multiple namespaces. This means that in scheme a name has a single meaning in any instant of time. In Common Lisp it depends. To do so, a name is considered a function only if it's in the first position and the others are variables. This can lead to several problems in the implementation of the language and especially the problem of ambiguity.
So to solve the ambiguity problem there are:

- `#'` that is used to indicate that a name is a function
- `funcall` that is used to call indirectly a function

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.

The problem is that when we are defining a function that contains free variables, there should be a way to store the values of those variables. In Pascal, the variables are stored in the stack. In C, every free variable should have a global scope. The thing is that to make something like this available we need to know for each function the environment where it was created. This can lead to indefinite expansion of the environment leading to memory problems.

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 ≤ i ≤ 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.

A problem that can be solved using non-deterministic computation is a problem where there are alternative ways to continue a computation and that can be solved using a backtracking approach.

Exam 7

What is reification

Reification is a meta system is creating an entity that represent the system. It's a pre-condition for reflection.

- What is the class of this instance? reification of classes
- What is the method of this instance? reification of methods
- What is the invocation chain? Reification of stack

In java there are some reflection capabilities. can and can't do?
In Java you can know the class of an object, the methods of a class, the fields of a class, the annotations of a class, and you can invoke methods at runtime. But you can't change the structure of a class at runtime or change the structure of a method at runtime.

5. (1.0) A linguagem Common Lisp permite classes com heranc¸a m´ultipla. Neste contexto, explique o que´e a lista de precedˆencias de classes e refira para que serve

In common lisp this list of precedences is built for a class C in order to make sure that the correct overloading of the methods is done correctly. In essence, the class precedence list ensures that methods defined in superclasses are appropriately inherited and overridden in subclasses, maintaining consistency and predictability in method resolution in the presence of multiple inheritance. It provides a clear hierarchy of class relationships, allowing for efficient method dispatch and ensuring that methods are invoked in the correct order when dealing with class hierarchies.

6. (1.0) The CLOS language specifies a set of meta-object protocols. What are they for? Choose one of these protocols and explain it.

Meta-object protocols serve to mediate the manipulation of objects through their corresponding meta-objects. For example, accessing a slot of an object is done through the slot-value function, but it is mediated by the generic function slot-value-using-class, which is specialized in the metaclass of the object. To implement different behavior for accessing a slot, we can define a new metaclass and specialize the generic function slot-value-using-class for that metaclass in order to implement the desired behavior.
For example, having an object Car with a slot color, we can define a metaclass CarMeta that specializes the slot-value-using-class function to implement a custom behavior for accessing the color slot of a Car object.

Exam 8

1. What is reflection and what is a reflective system?

Reflection is the ability to reason about the structure of a problem. A reflective system is a system that can reason about its own structure, so it's a system that has itself as object system.

2. What is a reflective architecture and what do we need for it?
A reflective architecture is a structure for a system that makes able the inspection of its parts. In particular, there is the need to represent the entity of the system in order to be able to inspect it.

3 Considering the various dimensions of reflection (introspection, interception, structural, behavioral), classify the following mechanisms:

(a) (0.5) Operation that traces function invocations, showing the arguments and the result each time the function is called.

It's behavioral introspection

(b) (0.5) Operation that allows determining the number of parameters of a function.
Structural introspection

(c) (0.5) Operation that allows resuming the execution of a program that was interrupted by an error produced by a function invocation, by imposing a return value to that invocation.

Behavioral interception

(d) (0.5) Operation that allows changing an instance to belong to a different class than the one used for its creation.

Structural interception

4. What are annotations used for in Javassist?
Annotations in Java are metadata that can be added to Java source code to provide additional information about classes, methods, variables, or other program elements. They serve various purposes, including configuration, documentation generation, and providing instructions for frameworks and libraries.

For Javassist, annotations can be particularly useful in bytecode manipulation. They allow developers to mark classes, methods, or fields with custom annotations to indicate that they should be modified dynamically at runtime.

5. The Java language allows for dynamic class loading, where classes can be loaded into memory at runtime. However, whenever a class is loaded, it is immediately associated with the class loader that loaded it. what is the problem that is solved usign this technique?

By associating each loaded class with its respective class loader, Java provides a mechanism for creating isolated class loading environments. This means that classes loaded by one class loader are kept separate from classes loaded by another class loader, even if they have the same fully qualified name. This helps prevent classpath conflicts and ensures that different versions of the same class can coexist peacefully within the same JVM.

Overall, the association of classes with their class loaders helps maintain modularity, flexibility, and version control within Java applications, allowing for more robust and maintainable software systems.

6. In contrast to Java, where methods belong to classes, in CLOS (Common Lisp Object System), methods are not directly associated with classes. Why?

The reason is multiple dispatch.

7. (2.0) Different programming languages adopt different approaches regarding the scope of names, which can be either lexical or dynamic.

(a) (1.0) Under what conditions can these two scopes produce different results? Explain.
In the presence of higher-order functions and functions with free variables. In lexical scoping, the value of a free variable is determined by the environment in which the function was defined, while in dynamic scoping, the value of a free variable is determined by the environment in which the function is called. This can lead to different results depending on the scope of the variable.

(b) (1.0) Explain the changes that need to be made in the implementation of a dynamic scope evaluator to make it perform lexical scope evaluation.
Functions would need to be associated with the environment where they were created. Function application would extend this environment (with the associations between formal and actual parameters) instead of the dynamic environment.

8. (2.0) Lazy evaluation, also known as delayed evaluation, is used by several modern programming languages, such as Haskell, O'Caml, and F#.

(a) (1.0) What is lazy evaluation? Explain.
Lazy evaluation, also known as delayed evaluation, is a programming technique where expressions are not evaluated until their results are actually needed. In other words, computations are deferred until they are required by other parts of the program. This approach allows for more efficient use of

resources by avoiding unnecessary computations and can enable the evaluation of infinite data structures.

(b) (1.0) How can you implement lazy evaluation in a metacircular evaluator? Explain. The metacircular evaluator should use lexical scope, and each time a non-primitive function is invoked, each of its arguments should be "wrapped" in a structure that associates the corresponding expression with the environment in which it was supposed to be evaluated. Whenever a primitive function is invoked, the wrapped arguments are "unwrapped" by evaluating the expression in the associated environment. To avoid multiple evaluations, it's beneficial for the "wrapper" to be manipulated to record the result of the initial evaluation.

Exam 9

1. What is reflection? What degrees of reflection do you know?
Reflectio the ability of a program to inspect and modify it's own structure and behavior during his run time. There are two degrees of reflection: introspection and intercession.

- introspection: Introspection is the ability of a system to inspect and analyze his structure and behavior and getting information at runtime
- interession: Intercession is the ability of a system to modify and change the structure and behvaior of the system at runtime

1.5: What is reification?
Reification is the process of making an abstract concept more concrete or real. In the context of programming languages, reification refers to the process of representing abstract concepts, such as classes, methods, and objects, as concrete entities that can be manipulated and inspected at runtime. Reification is a key feature of reflective systems, which allow programs to reason about and modify their own structure and behavior.

2. (1.0) In order to allow reflection, the Java language has reified several concepts. Name three of these concepts
Java has reflective capability and are possible because there is a reification for:

- classes
- methods
- fields

3. What is metaprogramming?

Meta-programming is a programming technique in which the programs are written in a way that permits them to alter the behavior and structure of another program.
One example are the **macros**, which are a particular kind of meta-programming that allows the programmer to generate new code starting from a base one. A metaprogram **manipulates** a data structure that represent another program.

4. (2.0) Some argue that Java language interfaces can be used to implement multiple inheritance. Do you agree? Explain your answer
Java language interfaces can indeed be utilized to achieve a form of multiple inheritance, but it's important to clarify the distinction between multiple inheritance and interface implementation.

Interface Implementation: In Java, a class can implement multiple interfaces. Interfaces in Java provide a way to achieve abstraction and multiple inheritance of type. By implementing multiple interfaces, a class can inherit the abstract methods defined in those interfaces. However, interfaces cannot contain implementation details, only method signatures.

Multiple Inheritance: Traditional multiple inheritance refers to a language feature where a class can inherit from more than one superclass. This can lead to the "diamond problem," where ambiguity arises if two superclasses of a class have a method with the same name. Java does not support multiple inheritance of implementation for classes, meaning a class cannot directly inherit from multiple classes.

In conclusion, while Java does not support multiple inheritance of implementation for classes, it does support multiple inheritance of type through interfaces. Therefore, in Java, interfaces can be used to achieve a form of multiple inheritance, albeit in a different sense than traditional multiple inheritance.

5. (2.0) In CLOS, what is the difference between a generic function, a primary method and an auxiliary method? method? Explain
CLOS - Common Lisp Object System
A generic function is a particular function that is particularized by different types of arguments and has different methods that implements it. To do so, there is the concept of axuliary and primary method.
A primary method is a method that actually returns a value and represent the main behavior, instead an auxuliary method is a method that alters the behavior of the primary method. The primary method is the method that is actually called when the generic function is invoked, while the auxiliary methods modify the behavior of the primary methods.

6. What is the gensym function? What it does and where it's used?
The gensym function exists in LISP and it's main goal is to return a new symbol everytime it's called. This is used particular in the creation of **macros** in metaprogramming and it's useful because it avoids the multiple evaluation of the same expression.

7. (1.0) Pascal permits that some functions can be defines inside other functions but not returned as a value. C permits to return functions as values but not to define functions inside other functions. What is the problem that explains these two different approaches to language design?

The problem is the scope of the variables. With the Pascal approach, the variables are stored in the stack, while in C the variables are stored in the heap. This can lead to memory problems and the indefinite expansion of the environment.

8. Some languages consider a specific order of operand evaluation for an operator (for example, from left to right), while others leave this order unspecified. Under what conditions do you consider one of these options preferable to the other?

The order of operand evaluation can be important when the operands have side effects. A side effect is an effect that is not the result of the main computation, but that can affect the result of the computation. For example, if the operands are function calls that modify a global variable, the order of evaluation can affect the final result. In this case, it's preferable to specify the order of evaluation to avoid unexpected behavior.

Exam 10

1. (1.0) Explain and exemplify the concept of computational meta-system

A computational meta system is a system that can reason about another system. Clearly speaking, it's a running program that analyzes and modifies another running program. An example can be a debugger, that runs over a program and can analyze his runtime behavior.

2. (1.0) Explain and exemplify the concept of reflective system.
A reflective system is a system that can reason and change it's behavior at runtime. An example can be a program that can inspect and modify its own structure and behavior during execution. An example can be a meta-circular evaluator.

3. (1.0) Explain and exemplify the concept of reification
The term reification defines a concept of **creating an entity of a system** making possible the access to the structure. More specific, the creation of an entity that resemble the object system. An example is the reification of classes in java. You can get the class of an object and inspect it.

4. (1.0) There are many languages providing introspection mechanisms but only some of them also provide intercession mechanisms. Why? Explain.

Because it is harder to define the semantics of a program that can change while it runs, it is difficult to debug a program whose source code is not stable, and it is difficult to optimize a program that does not has a static form.

5. (1.0) The Lisp language invented the backquote syntax. For what purpose? Explain.
Backquote in LISP is a particular operator that don't evaluate the expression passed as argument. Instead, to actually evaluate the expression you can use the comma operator that given something inside the expression, it evaluates it. For example:
`1 2 ,(3+1) 4` will return `1 2 4 4`
It simplifies metaprogramming to generate new code parts at runtime.

6. (1.0) What is the purpose of Javassist? Explain
Javassist is a java extension that permtis to add intercession capabilities to java. It allows to inspect and modify the structure of a program at runtime. It's used in metaprogramming, debugging, and other tasks that require dynamic code generation and manipulation.

8. (1.0) Regarding method calls, Java uses dynamic dispatch for the receiver and static dispatch for the arguments. What are the advantages and disadvantages of this approach when compared to multiple dispatch? Explain
Performance
Unexpected results

11. (1.0) In the context of CLOS, explain the concept of effective method.
A generic function can have many different methods. Given specific arguments to the generic functions, the applicable methods are selec- ted, sorted, and finally combined in the effective method, that is, the method that is actually applied to those arguments.

12. (1.0) CLOS provides the concept of metaclass. Explain its purpose and responsibilities.
The metaclass of an object is the class of the class of the object.
The MetaClass Responsibilities are:

- Determines the inheritance process that is used by the classes that are its instances.
- Determines the representation of the instances of the classes that are its instances.
- Determines the access to the slots of the instances of the classes that are its instances.

14. (1.0) The meta-circular evaluator implemented in this course provided macros. Explain this concept.
It makes programmer able to extend a program. It's a transformation starting from an expression and generating another expression. It's used in metaprogramming, where you write code that generates code.

15. (1.0) What is the difference between direct style and continuation-passing style? Explain.
Direct style it's the standard way of programming in an imperative language. It's a style where the program is written in a way that the evaluation of an expression is done in a direct way
Continuation passing style it's based on continuation, that's what it remains to be done after the evaluation of some expressions.
In continuation passing style you need to pass the continuation as an argument to all the function calls.

16. (1.0) Suppose that you invented a tree-based data structure and you want to provide an iterator for its leaves. Do you prefer to provide an internal iterator or an external interator? Why?

An Internal Iterator is a higher-order function that applies a function to each element of a collection.
An External Iterator is an accessor of the current element of a collection, and, potentially, of the next one.

Looks like the internal iterator is the correct choice for this problem because you don;t need to save the value.

18. (1.0) Julia is a recently proposed programming language that supports higher-order functions. As an example, the following fragment defines the map of a function over a list.

```
map(f, list::Nil) = list
map(f, list::Cons) = cons(f(head(list)), map(f, tail(list)))
```

Do you think Julia provides different namespaces for functions and variables or, instead, a common namespace? Explain.
Given that the parameter f is being used in function position, Julia must use a common namespace for functions and variables.

Exam 11

I am skipping some repetitive af questions

3. (1.0) Self-modification is a extremely powerful capability that is provided by certain programming languages. However, this capability also causes certain problems. Discuss two of those problems.

One of the problems is that compilation becomes very difficult because the program does not have a stable form.
Another problem is that debugging is also greatly hindered by the fact that the program where the error was triggered may already be very different from the one written by the programmer.

10. (b) The presence of higher-order functions in a dynamically scoped language can lead to two serious problems. Which ones?

The downward funarg problem occurs when a function is passed as an argument. In dynamically scoped languages, the function may reference variables from its caller's scope. However, if that caller's scope changes before the function is invoked, the function may behave unexpectedly due to accessing outdated variables.

The upward funarg problem arises when a function is returned as a result. Similar to the downward funarg problem, dynamically scoped languages allow functions to capture variables from their enclosing scope. If a function is returned from another function and executed in a different scope, it may unintentionally access variables from its original scope, leading to unexpected behavior.

These problems occur under dynamic scoping conditions, where variable bindings are determined by the calling context rather than lexical structure. Dynamic scoping can make it challenging to reason about variable scope and behavior, especially when higher-order functions are involved.

(c) What is a macro.
A macro is a function that takes syntactic forms as arguments and computes a syntactic form as a result, which is evaluated in place of the macro invocation.

The purpose of a macro is to define syntactic transformations, allowing programmers to create new control structures or language constructs using existing ones. For example, macros enable the definition of custom control flow structures that may not be directly supported by the language syntax. By providing a mechanism for metaprogramming, macros enhance the expressiveness and flexibility of a programming language, enabling developers to create domain-specific abstractions and improve code readability and maintainability.

Exam 12

5. (1.0) Recent versions of the Java language provide annotations. Explain this feature and discuss its usefulness for Javassist.
Annotations in Java are a form of metadata that can be added to Java source code elements such as classes, methods, variables, and packages. They provide additional information about the code to the

compiler or runtime environment, which can be used by tools or frameworks to generate code, perform static analysis, or configure runtime behavior.
Javassist is a Java library that provides a means of manipulating Java bytecode at runtime. It allows developers to dynamically modify classes, create new classes, and inject code into existing classes. Annotations play a crucial role in Javassist by providing a mechanism for identifying which classes, methods, or fields to modify or enhance.

In this way there is no need to over engineer the code to find the methods that needs to be modified or enhanced, and makes the code more readable and maintainable.

7. (1.0) In many object-oriented languages, every value is a member of a class. In some of those languages, a class is also a value. What consequences do you extract from these two ideas? Explain.

If every value is a member of a class, it means all data in the language is represented as objects, promoting a consistent object-oriented model.

If a class itself is also a value, it means classes can be treated as first-class entities, allowing for powerful metaprogramming capabilities like creating classes dynamically at runtime or passing classes as arguments to methods.

These two ideas promote a more uniform and flexible object-oriented programming model, but they can also add complexity and overhead compared to languages where classes are not first-class values.

9. (1.0) The Java language does not provide multiple dispatch. What alternatives to multiple dispatch can you use in Java? Explain them.

- Double Dispatching: This technique involves creating a separate method in the receiver object that dispatches the request to another method in the argument object. This second method then performs the desired operation based on the types of both objects. This approach can be cumbersome and can lead to code duplication
- Java's reflection API can be used to dynamically inspect the types of the arguments and dispatch the appropriate method at runtime. This approach can be flexible but can also be less efficient and more complex than static dispatching.
  Instance specialization in CLOS (Common Lisp Object System) can be useful in situations where you need to define specialized behavior for specific instances of a class, rather than just based on the types of the arguments.

10. (1.0) In CLOS, generic functions can be specialized not only on the types of the arguments, but also on specific arguments. This is known as instance specialization. Describe a situation

where instance specialization is useful.

Instance specialization in CLOS (Common Lisp Object System) can be useful in situations where you need to define specialized behavior for specific instances of a class, rather than just based on the types of the arguments.

One scenario where instance specialization can be beneficial is when dealing with objects that represent specific entities or resources, and you need to define custom behavior or operations for those specific instances.

Another use case for instance specialization could be in a gaming environment, where you might have a move generic function that handles the movement of game characters. You could define a specialized move method for a specific instance of a character, such as the player's character, to apply special rules or behaviors only for that instance.

Exam 13

2. (1.0) Explain the concept of short-circuit evaluation. Which operators are usually implemented using this form of evaluation? Why?
Short-circuit evlauation is a form of evaluation where the evaluation of an expression stops as soon as the result is determined. This is particularly useful for logical operators like AND and OR, where the second operand may not need to be evaluated if the result can be determined from the first operand For example, imagine having to divide a number by zero. In this case, if you already know that the first operand is zero, you don't need to evaluate the second operand, since the result will be zero anyway.

3. (2.0) Different programming languages adopt different strategies regarding scoping. The two main approaches are named lexical scope and dynamic scope.

What are the main differences between these two scoping strategies?

- lexical scoping: the scope of a variable is determined by its location in the source code. Variables are resolved based on the program's structure, and the scope of a variable is determined by where it is declared in the code.
- dynamic scoping: the scope of a variable is determined by the order of function calls at runtime. Variables are resolved based on the order in which functions are called, and the scope of a variable is determined by the call stack.

(a) (1.0) Under what circumstances would these two scoping strategies produce different results?

When we are deadling with high order functions, that are functions that can receive other functions as arguments or return functions as results. In this case, the scope of the variables can be different depending on the scoping strategy used. Another case is when we are dealing with free variables, that are variables that are not defined in the function but are used in the function. In this case, the scope of the variables can be different depending on the scoping strategy used.

(b) (1.0) What changes must be done in a meta-circular evaluator that provides dynamic scope so that it provides lexical scope instead?

Functions become associated with the environment where they were created. The application of functions extends this environment (with the associations between formal and actual parameters) instead of the dynamic environment.

5. (1.0) Some programming languages specify the evaluation order of the argument expressions of a function call, while other programming languages leave that order unspecified. Under what circumstances is one of the options preferable to the other? Explain.

When a language promotes the use of side effects, it is generally preferable for the programmer to be able to rely on a left-to-right evaluation order.

7. (1.0) Consider a meta-circular evaluator that provides functions and macros. Describe the diffe- rences between the implementation of function calls and macro calls.

Function calls evaluate the argument expressions, bind parameters to the corresponding values, and evaluate the body of the function.
Macro calls do not evaluate the argument expressions, bind parameters to these non-evaluated expressions, evaluate the body of the macro, and evaluate the result of the previous evaluation

Function call: evaluate arguments -> bind parameters -> evaluate body
Macro call: don;t evaluate expr -> bind parameters -> evaluate body -> evaluate result