The Cerium Reference Manual

Contents

1. Introduction

*** Note that Cerium is a work in progress. I still don't have the code generator built and I have language features to add. This means that this document is a combination of things that are already built and some things that are still yet to be developed. For example, when I say that this is a language that runs on the JVM, it doesn't yet run on the JVM, since I still have to build the code generator. ***

This manual describes the programming language Cerium. The purpose of this document is to describe the language features of Cerium and introduce the basic syntax and semantics of the language.

Cerium is an OOP, statically typed language that runs on the JVM.

Cerium programs consist of a set of .cerium files, where one or more classes are defined. Every class defines a type. Each class encapsulates the members and methods of that class.

Inheritance allows new types to extend the behavior of existing types. Cerium uses the *extends* keyword to specify the superclass, the same as in Java.

2. Getting Started

The best way to get a feel for the Cerium language is to look at an example .cerium file. The project contains a number of .cerium files in the source-code directory within the project. The language itself resembles a variety of languages in different ways. The goal is to keep it aligned with Java syntax where it makes sense, but remove Java's verbosity where possible. At the same time, Cerium tries not to borrow too much from functional languages in terms of syntax, since it's not functional and doesn't want to pretend to be something it's not.

Within the project, there is a Compiler.java file. This is the class that actually performs the lexing, parsing, semantic analysis, and (eventually) the code generation routines. This class contains a main method, that takes a .cerium file as input (I currently hard-code a reference to a single .cerium file) and performs these actions. Note, that it currently only outputs a description of the annotated AST. Instead of writing this information to the console, I'll perform code

generation, once that part has been developed. Also, I'll need to compile multiple .cerium files. Right now, I'm only doing one file at a time, as this project is still in its infancy.

3. Classes

A class in Cerium is defined by the keyword *class* followed by the name of the class. The class body is enclosed in braces {}. An empty class definition looks like this:

```
class Person {
}
```

You can extend a class in Cerium using the *extends* keyword and the name of the class you want to extend. You can only extend one class in Cerium (i.e. there is no multiple inheritance).

```
class Employee extends Person {
}
```

When extending a class, the child class inherits all non-private properties and methods, as you've come to expect from other general purpose programming languages.

4. Types

Cerium has the following built-in types:

int
float
void
char
boolean

When computing expressions, Cerium will auto-promote types appropriately. For example, 3 + 2.5 will evaluate to a float type. You'll get an error, however, if you try to assign that value to an int type, since you lose the decimal point precision when you try to store the result in an int type.

5. Attributes

Class attributes in Cerium are listed within the body of the class and outside any class methods, using the following format:

<access-modifier> <attribute-name> : <attribute-type> ;

Here's an example:

```
class Employee : Person {
   // class attributes
   age : int;
   private hourlyPayRate : float;
};
```

This is a spot where Cerium deviates from Java and defines the type of the attribute after the attribute name, instead of before the attribute name (like Scala).

Note that class attributes do not need to be defined at the top of the class. Most developers prefer to keep their attributes at the top of the class so that they're easier to find, but it's not required.

Attributes are public by default, just like in Scala. You can specify *private* or *protected* if you want to change the access of the attribute. *public* is not a keyword in Cerium, since that's the default access.

6. Methods

Class methods are defined inside of a class and are defined with the following format:

<method-name> (<parameters>) : <return-type> { <method-body> }

Here's an example method named "foo" in the Employee class that does not return a value (hence the void return type).

```
class Employee extends Person {
   // class attributes
   age : int;
   private hourlyPayRate : float;

   foo() : void {
   }
}
```

To return a value from a method, you use the *return* keyword. Here's an example of foo being modified to return 10% of the employee's hourly pay rate.

```
class Employee < Person {
   // class attributes
   age : int;
   private hourlyPayRate : float;

   foo() : float {
      return hourlyPayRate * .10;
   }
}
```

7. Expressions

Todo …