PHAS3459: Scientific Programming Using Object Oriented Languages

Module 10: The Differences Between Java and C++

Dr. B. Waugh and Dr. S. Jolly
Department of Physics & Astronomy, UCL

Contents

1. Aims of Module 10

The aims of this module are:

- to give you a brief overview of the history of C++;
- to give you a flavour of some of the key differences between Java and C++;
- to provide you with some links to resources should you wish to learn more about C++ and its relationship to Java.

The content of this module is not examinable!

You will not be able to implement the code examples here using the Eclipse Java IDE!

2. C++: Background

C++ was created in the early 1980's as an extension of the existing programming language "C" with the object-oriented features of "SIMULA-67". Since then it has undergone a number of changes and improvements; the language specification is now very stable and is codified by the American National Standards Institute (ANSI). C is a very popular programming language with large existing libraries of utilities that are particularly suitable for "low-level" applications — for example controlling computer hardware and peripherals. C++ is backwards compatible with C and so enjoys these same advantages, while at the same time having a highly developed object-oriented structure suitable for "higher-level" applications.

Most modern operating systems are largely written in C/C++ and C++ is the language of choice for many scientific applications. For example particle physicists particularly like C++ since their range of applications include low-level tasks (e.g. control of their detection apparatus) as well as higher-level tasks (e.g. data analysis and simulation).

In terms of performance, a well-designed C++ program will often run faster than a corresponding Java program since there is no overhead of running a JVM (Java Virtual Machine) to interpret the byte-code. Since C++ is "closer to the hardware" it's also possible to create highly optimised specialised applications in C++ that would be harder to implement in Java.

3. Compiling and Running C++ Programs

C++ is a conventional compiled language. A compiler converts a C++ program into a file containing low-level instructions that are understood by the computer's CPU. If the program you are writing calls functions defined elsewhere, then the corresponding "libraries" must be "linked" in order to generate the final executable.

These steps are illustrated in the following example of the simplest possible C++ program:

```
#include <iostream>

int main(int argc, char *argv[]) {

  // This is similar to System.out.println("...") in Java :
  std::cout << "Hello World !" << std::endl;

  return 0;
}
```

Then, for example, on a computer running a UNIX/Linux-variant operating system and with a C++ compiler called "`g++`" installed, one could do the following:

- Compile the program: "**`g++ -c helloWorld.cc -o helloWorld.o`**". The compiler reads the code above in the file `helloWorld.cc` and creates the "object file" `helloWorld.o` containing the low-level computer instructions.
- Link the program: "**`g++ helloWorld.o -o helloWorld.exe`**". At this point all the other object files and libraries needed to run the program must be specified. In this case only a single object file is required to build the executable called `helloWorld.exe`.
- Run the program: "**`helloWorld.exe`**". The message "`Hello World !`" will be printed to the screen.

The object files that are created by the compiler are in some respects the analogue of the byte-code "class" files produced when you compile Java code. Crucially, however, they are not portable in the same way that byte-code files are. Object files and executables produced on one particular platform (combination of hardware and operating system) will generally only work on identically configured computers.

4. Some Important Differences Between Java and C++

Java and C++ are syntactically quite similar. For example most of the techniques you have learned for manipulating built-in types and controlling algorithm flow will translate directly into C++. With your knowledge of Java, you will probably be able to read a well written C++ program and have a good idea what it is doing.

However there are important differences between Java and C++, some of which are enumerated here. This is *far* from being a complete list.

4.1 Non-Class Code

One consequence of C++'s origins is that it's possible to write an entirely non-object oriented (sometimes called "procedural") program. In particular functions can be defined that belong to no particular object and which can be called directly from "`main`", or from any other function. This is in contrast to Java, for which every function, including "`main`", must be a method of a class. In certain circumstances the flexibility of C++ in this respect is an advantage, but in general the discipline involved in having to organise all Java code into suitable classes helps encourage better program design.

4.2 Pointers and References

In Java variables are references to instantiated objects, and the same concept exists in C++. However in C++ a slightly different way of referring to objects exists through the concept of “pointers”. Pointers literally point to the “memory-address” or the location in physical memory occupied by an object.

Additional manipulations are possible with pointers that are impossible with references. For example “pointer-arithmetic” makes it possible to literally navigate through a computer's memory byte-by-byte. Although this would not be recommended in ordinary programs, it can be useful in constructing highly efficient low-level applications (again, this language feature reflects the C heritage of C++).

4.3 Memory Management

In Module 5 you learned about Java's built-in “garbage-collection”. Java will delete objects that can no longer be referred to in a running program (i.e. all references to that object have been lost), and free-up the memory previously occupied by those objects.

C++ has no built-in garbage-collection. What this means is that the programmer must take care to explicitly delete objects when they will no longer be used. Failure to do this can result in objects with no existing valid pointers continuing to occupy memory, called a “memory-leak”. Memory leaks are a common problem in poorly-written C++ program; when the memory leak is sufficient to exhaust the available memory on the computer it will cause the program to crash. Functionality similar to Java's garbage-collection can be replicated in C++ using specialised techniques, or using existing libraries developed for this purpose.

4.4 Destructors

In order to aid object deletion, objects have “destructors” defined as well as “constructors”. Roughly speaking, every “`new`” must be accompanied by a corresponding “`delete`” as illustrated here:

```
#include <iostream>

class Planet {

public:

  Planet(std::string name) {
    _name = name;
    std::cout << "A planet is born, with name "
              << _name << std::endl;
  }

  ~Planet() {
    // This is Planet's destructor. This method will be
    // called when a Planet object is deleted :
    std::cout << "A planet dies, with name "
              << _name << std::endl;
  }

  std::string name() {return _name;}

private:

  std::string _name;

};

int main(int argc, char *argv[]) {

  // "earth" is a pointer to an object of type "Planet" :
  Planet* earth = new Planet("Earth");
  std::cout << "Welcome from " << earth->name() << std::endl;
  delete earth;

  return 0;
}
```

4.5 Inheritance

Although the syntax is a little bit different, C++ supports basic inheritance in a very similar way to Java. One major difference is that C++ does not directly contain the concept of "interfaces". Instead, in C++ one can defined "purely-virtual" base-classes which, when inherited, force the sub-class to implement certain functionality in a similar way to Java interfaces. In C++ there are no restrictions on the number of super-classes that a sub-class can directly inherit from. This can lead to substantial confusion in poorly-designed object-oriented programs.

4.6 Java's "`Object`" Class

In Java all objects inherit from the "`Object`" base-class. In C++ there is no such structure and, indeed, as pointed out in the section on Non-Class Code, it's entirely possible to write a complicated C++ program involving no objects at all.

5. C++ Standard Library

Roughly equivalent to the Java API with which you are now familiar is the "C++ Standard Library". It contains generic containers (directly analogous to and often identically named — e.g. "`Vector`" — to those available in Java) as well as commonly used string and I/O classes and utilities. Importantly, the scope of the C++ Standard Library is considerably more restricted than that of the Java API so, for example, there exists no equivalent standard set of graphics utilities.

There are huge numbers of publicly available C++ code libraries for performing more specialised tasks. Even more so than is the case with Java, the first step in a complex C++ programming exercise should be a review of already existing and available code that might be of use for the task at hand.

6. Resources

There are an almost infinite number of resources available if you want to learn more about C++. As is also the case with Java, there are a number of good books in print; for a list of reviews see:

http://www.accu.org/index.php/book_reviews

In terms of comparisons between C++ and Java, the following may be useful:

Wikipedia: http://en.wikipedia.org/wiki/Comparison_of_Java_and_C++

Moving from Java to C++:

http://www.horstmann.com/ccj2/ccjapp3.html

Similarities and Differences between Java and C++:

http://www.dickbaldwin.com/java/Java008.htm

7. Summary

In this module you were given the briefest of introductions to the C++ programming language and some of the key similarities and differences between C++ and Java. If you need to learn more about C++ you can use some of the links to resources mentioned above.