Erich's Java cheat sheet for C++ programmers

©Erich Kaltofen
kaltofen@math.ncsu.edu

October 29, 2013

C++	**Java**
assignment `operator=`	cannot be user-defined for a class and performs assignment of a reference to the instance of the class (see also reference types)
`basic_string`	`String` and `StringBuffer`
`bool`	`boolean`
`char`	`byte`
`const` variables/data members	`final` variables/fields
copy constructor	no default; one implements the interface `Cloneable` by the method `Object clone()`, which can be an abstract (in C++ notion: virtual) method
data members	fields, so-called *instance variables* (a term borrowed from Smalltalk)
`delete`	does not exist; all unreferenced memory is garbage collected
derived classes	subclasses; the keyword `extends` replaces C++'s colon.
destructors ˜*Class*	`protected void finalize()`; note, however, that these are used for freeing resources **other than memory** and are therefore rarely needed
exceptions, `try`, `catch`, `throw`, `std:exception`	same concept; Java adds a keyword `throws` that is used to declare the exceptions a method throws; the hierarchy of exceptions is rooted in `java.lang.Exception`; a `finally` block is introduced to contain all common clean-up code.
`extern "C"` functions	`native` methods
functions	do not exist; `static` methods ("class methods") are used
`#include`	does not exist; the paths to the files are known and can be made know in the `CLASSPATH` environment variable

C++	**Java**
input/output: `istream& operator>>`, `ostream& operator<<`	`System.in` and `System.out` are the streams; Java has number formatting tools in `java.lang.Number` and `java.text.Format.NumberFormat`
`main(int argc, char* argv[])`	`public static void main(String [] args)` within a `public` class
member functions	methods
multiple inheritance	does not exist; however, interfaces provide a weak form of multiple inheritance.
namespaces	packages
`namespace` *Namespace*`{`...`}`	`package`*Package*`;` which must appear as the first line in the file
nested (member, inner) classes	Java 1.1 has `static` ("top-level") and non-static ("member") inner classes, as well as local classes and anonymous classes. Member classes can refer to the members of the outer class and to *OuterClass*`.this`; they cannot have the name of an outer class and cannot declare `static` members.
`new` *Class*`(`...`)`	`new` *Class*`(`...`)`, which returns a reference to the created object
`NULL` (the 0 pointer value) and the type `void*`	`null` in Java is a keyword and represents an uninitialized reference
overloaded operators	do not exist; however, methods can be overloaded. This may be a major shortcoming of Java, as one cannot revise old Java code by redefining the operators used (cf. MITMatlab)
passing arguments to base class constructor	place the statement `super(`...`);` as the first statement in the subclass's constructor
`public`, `private`, `protected` modifiers	similar as in C++; visibility of classes and nested classes can be also restricted; there are no friends, but within the same package protected members are visible
purely `virtual` member functions	`abstract` methods; the enclosing class must also be declared `abstract`
reference types *Type*`&`	all Java types except scalar primitive types are reference types; note that the method `void swap(`*T* `a,` *T* `b) {`*T* `t; t = a; a = b; b = t;}` does nothing to its arguments.
scope resolution, operator `::`	does not exist; methods must be defined inside the class declaration. If a base class field is to be explicitly referred, one uses typecasting: `((`*Baseclass*`)`*Variable*`)`.*Member*; a direct base class member can be referred to by `super`.*Member*; typecasting has no effect on methods (see `virtual` member functions).

C++	**Java**
`static` data members	`static` fields, so-called *c*lass variables; they are accessed by *Class*`.`*Field* rather than the C++ *Variable*`.`*Member*; they can be initialized by =...; within the class definition and need not be declared outside like C++ static data members.
`static` member functions	`static` methods, so-called *class methods*; they are defined within the class declaration, unlike in C++.
`this`	`this`, which is a reference to the object and has the type of the class, not a pointer; note that the call `this(...);` as the first statement in a constructor invokes a constructor call for the matching argument types.
traits	marker interfaces
`type_id`	`instanceof`; this is an operator returning a `boolean`, not a "`type_info`" as in C++.
`using namespace` *Package*;	`import` *Package*`.*`;
`virtual` member functions	in Java, all methods use dynamic method lookup and therefore are be default virtual. There is no way to explicity call an overridden base class method, but overwriting can be prevented by declaring a method `final`.
`wchar_t`	`char`
wide character stream `wostream`	`PrintWriter` replaces `PrintStream` that cannot hold unicode; the constructor of `PrintStream` has been deprecated in Java 1.1, but `System.out` is not.

Java concepts missing in C++	
abstract windows toolkit AWT	standard library for building a GUI
concatenation of strings by + operator	
documentation comments	can be processed (e.g., by `javadoc`) for automatic online documentation
`final` methods	those cannot be overridden by a subclass
interfaces	are used to denote abstract classes without any method of their own. They can have `static final` fields. One class can implement several interfaces, but it must implement the abstract methods of each interface.
reflection	allows the inspection of a class (which arguments does which member take? etc.); this is critical for plug-and-play design, such as a Java bean
right shift operator with zero extension >>>	
serialization	C++ requires the programmer to implement object serialization member functions

Java concepts missing in C++
sockets
threads

C++ concepts missing in Java

C++ concepts missing in Java	
`const` member functions	do not exist; `final` methods cannot be overridden by subclasses
`friend` classes, functions	do not exist; however, `protected` members are visible within the same package
`goto`	is a reserved work in Java, but is not supported by the language; however `break` and `continue` statements can give a statement label
multiple inheritance	`virtual` base classes seem unachievable by using interfaces
`new(`*Pointer*`)` *Type*`(`*...*`);` *Pointer*`->~`*Type*`();`	this is C++'s explicit memory allocation mechanism. In Java, all memory is managed by the VM and garbage collection is automatic. Thus, in C++, a garbage collector can be implemented, while in Java a memory manager cannot.¶
pointer types *Type*∗	do not exist; actually, since Java has only reference types, all variables are some kind of pointers and the = operator behaves like a pointer assignment
pointer to function, member	not a serious restriction, as one may encapsulate a function in a function object
standard template library STL	`java.util.Vector` provides an expandable vector. Java 1.2 provides `Collections`, which are essentially C++ STL containers, but many of the members are renamed. Note that `List` is a scrollable list in the AWT. There are third-party vendor container packages: See `http://reality.sgi.com/austern_mti/java/index.html`, `http://www.objectspace.com/developers/jgl/downloads/index.html`§
templates	there is a the GJ compiler `http://www.cs.bell-labs.com/~wadler/pizza/gj/`.§ C++'s template expansion mechanism is a full-fledged programming language and has been used for compiler optimization task (e.g., in the Blitz++ matrix library)
`typedef`	asside as a shorthand, `typedef`s can be encapsulated in a class scope to provide a generic type; they function as assignments in template meta-programming.
¶Laurent Bernardin points out that this isn't exactly true: place all objects on arrays/lists for reuse	
§These references were provided by Thierry Gautier	