Appendix A: From C++ to Java

Preprocessor

Java does not use preprocessor directives. The `import` statement is used to access library classes and packages.

Primitive data types, variables, and constants

`int`, `long`, `short`, `float`, and `double` are the same, but their sizes do not depend on the platform. There are no `unsigned` types in Java. `char` takes two bytes but is used the same way as in C++. `bool` is called `boolean`. The keyword `final` is used for `const;` also, `final` fields may get their values assigned in constructors. Literal numeric constants, character constants, and literal strings are written the same way (except the hex escape sequence in characters). There is no `enum` in Java.

Fields (data members of classes) and array elements of numeric data types are by default initialized to 0; `boolean` fields and array elements are by default initialized to `false`, and references to `null`.

Arithmetic, relational and logical operators

The assignment, arithmetic, compound assignment, and increment/decrement operators are the same. A C-style cast operator is used, as in `(double)k`.

Relational and logical operators are the same, but logical operators apply only to the `boolean` type operands. Short-circuit evaluation is used.

Control statements

`if-else`, `switch`, `while`, `for`, `do-while`, `break`, `continue`, and `return` are the same.

Functions (methods)

All functions (called "methods" in OOP) belong to classes — there are no free-standing methods. This applies to `main`, too — in a stand-alone application it must be a

```
 public static void main(String[] args)
```

in one of the classes. (An applet does not have a `main` — applet's `init` method is called instead.)

All primitive data types are always passed to methods by value. All objects (including strings, arrays, and programmer-defined types) are always passed to methods as references and returned from methods as references. There is no special syntax for that.

Overloaded functions and operators

Overloaded methods work the same way in Java as in C++. No overloaded operators in Java (but there are built-in + and += operators for concatenating strings).

No templates. No inline functions.

Pointers and references

No pointers in Java, only references. There is no special syntax for references: an object is always declared and accessed through a reference. An uninitialized reference is equal to `null` (a reserved word). The same "dot" notation as in C++ is used for accessing data fields and calling an object's methods.

An object (including arrays, strings, programmer-defined types) must be initialized using the `new` operator before it can be used. Exceptions are literal strings and arrays initialized with values — these are allocated automatically. For example:

```
 int points[] = {1, 2, 3, 4, 5, 6};
```

A reference to an object can be also returned by a method that creates the object. (Many methods from standard packages return references to objects.)

There is no `delete` in Java — automatic "garbage collection" is used.

Classes

There are no `structs` in Java. Only one public class can be defined per source file. No semicolon after the closing brace. Each class's field or method must be designated `public`, `private`, or `protected` individually. A destructor is `void finalize()`, but destructors are rarely used because the garbage collector releases memory automatically.

Inheritance is indicated by the keyword `extends`. There is only one ("public") type of inheritance in Java. No initializer lists in Java: the base class's constructor can be called explicitly using the keyword `super`. `super` is also used to call the base class's methods.

`this` is a reference, not a pointer (e.g., `return this`, not `return *this`).

Static methods are called and static fields are accessed using the class's name with a dot as a prefix (e.g., `Math.sqrt(x)`, `Color.red`).

There are no `friend` functions or classes in Java, but a private class can be embedded in a public class. The keyword `abstract` is used instead of `virtual`.

Pitfalls

1. The assignment operator applies to references to objects, not objects themselves. For example, in

   ```
   Employee emp1 = new Employee("Geena");
   Employee emp2 = emp1;
   ```

   `emp2` refers to the same object as `emp1`, not to a copy of `emp1`. You need to provide a copy constructor or a `clone` method for your class to make copies of objects. For example:

   ```
   Employee emp2 = new Employee(emp1);
   ```

   or

   ```
   Employee emp3 = (Employee)emp1.clone();
   ```

2. Relational operators applied to objects, as in

```
    if (str1 < str2) ...
    if (obj1 == obj2) ...
```

   compare references to (addresses of) objects, not their values. Use special methods `equals` or `compareTo` for strings.

3. Unfortunately, Java's bit-wise `&` and `|` operators may be used with `boolean` variables and expressions, but they do not recognize short-circuit evaluation. It is best to avoid them when working with `boolean` expressions.