C++ vs. Java Syntax Difference

	C++	Java

Main function / method	int main(int argc, char* argv[]) {
	    …
	}
		class Program {
		    public static void main(String[] args) {
		        …
		    }
		}

Comments	/* Comments in paragraph */ // Single line comment	/* Comments in paragraph */ // Single line comment /** Comments for Javadoc */
Primitive Data Types	short, int, long float, double, char bool (with signed and unsigned modifier)	byte, short, int, long float, double char boolean

Arrays	• int arr[10];
	• int* arr;
	   arr = new int[10];
	   delete [] arr;
	
	• No array bound checking
		• No equivalent
		• int[] arr;
		   arr = new int[10];
		   // Garbage collection
		   // delete is not required
		• Array bounds are always checked
		• A length instance variable is available in array to tell how many elements are there

Arithmetic, Relational, Logical Operators & Control Constructs	• Arithmetic: +, -, *, /, %, ++, --, +=, -=, *=, /=, %= • Relational: >, >=, <, <=, ==, != • Logical: &&, ||, !	• Arithmetic: +, -, *, /, %, ++, --, +=, -=, *=, /=, %= • Relational: >, >=, <, <=, ==, != • Logical: &&, ||, !

Control Constructs	• Branching: if-else, switch, conditional operator (i.e. ? :)
	• Looping: for, while, do-while
		• Branching: if-else, switch, conditional operator (i.e. ? :)
		• Looping: for, while, do-while, for-each
		for(<data type> <variable> :
		    <array> | <collection>) {
		    …
		}

Conditional Expressions	Can be bool or integral	Can only be boolean
Variable Instantiation	int* a = new int;	No equivalent

Input	#include <iostream>
	using namespace std;
	…
	int a;
	cin >> a;
		import java.util.Scanner;
		…
		
		int a;
		Scanner sc = new Scanner(System.in);
		a = sc.nextInt();
		sc.close();

Output	#include <iostream>
	using namespace std;
	…
	int a = 10;
	cout << a;
	cout << a << endl;
		int a = 10;
		System.out.print(a);
		System.out.println(a);
		// Data are concatenated using operator +

Class	class NewType {
	int x;
	int y;
	public: // Modifier for group of members
	NewType() : x(0) { // With MIL
	… }
	int func(int i) { … }
	void constFunc() const {
	// const member function
	…
	}
	};
		class NewType {
		private int x; // Initialized to 0 by default
		private int y = 0; // Initialize at declaration
		// No MIL
		// Each method with access modifier
		public NewType() { … }
		public int func(int i) { … }
		public void constFunc() {
		// No const member function
		}
		}

Separation of Class Definition & Implementation	Allowed - Class definition in .h file - Class implementation in .cpp file	Not allowed Class definition and implementation should all be in the same file

Object Instantiation	NewType a; // Create an object
	
	NewType* p = new NewType;
		NewType a; // Only create a reference,
		// no object is created
		// Always use new and allocate on the heap
		// Also, need parenthesis for constructor
		NewType p = new NewType();

Pointer vs. Reference	NewType* p;
	p = new NewType;
		NewType p;
		p = new NewType(); // () is needed

Const-ness	• const int x = 8;
	• const NewType* p = new NewType;
	p->x = 5; // ILLEGAL
	p = new NewType; // LEGAL
	• NewType* const p = new NewType;
	p = new NewType; // ILLEGAL
	p->x = 10; // LEGAL
		• final int x = 8;
		• final NewType p = new NewType();
		p.x = 5; // LEGAL
		p = new NewType(); // ILLEGAL
		• final NewType p = new NewType();
		p = new NewType(); // ILLEGAL
		p.x = 10; // LEGAL

NULL vs. null	int* p = NULL;	NewType p = null;
this Keyword	A pointer that points to the object whose member function is being invoked this-> OR (*this).	A reference that references to the object whose method is being invoked this.
Object Copying	NewType b = a;	NewType q = p.clone();

Data Member / Instance Variable Access	a.x = 10;
	p->x = 10;
		p.x = 10;

Member Function / Method Access	a.func(5);
	p->func(5);
		p.func(5);

Inheritance	class A {
	    private:
	        int a;
	        double b;
	    public:
	        A(int a, double b) {
	            this->a = a; this->b = b;
	        } …
	};
	
	class B : public A { // Use colon syntax
	    private: int c;
	    public:
	        B(int a, double b, int c) : A(a, b) {
	            this->c = c;
	        } …
	};
	Allow multiple inheritance
		class A {
		    private int a;
		    private double b;
		    public A(int a, double b) {
		        this.a = a; this.b = b;
		    }
		    …
		}
		
		class B extends A { // Use "extends" keyword
		    private int c;
		    public B(int a, double b, int c) {
		        super(a, b);
		        this.c = c;
		    } …
		}
		Only allow single inheritance

Virtual functions / methods	class C {
	    public:
	        virtual int func() { … }
	};
		class C {
		    // Methods are virtual by default,
		    // use final to prevent overriding
		    public int func() { … }
		}

Collections	vector<int> v;
	for(vector<int>::iterator it = v.begin();
	      it != v.end();
	      ++it) {
	  …
	}
		ArrayList<int> arrayList = new ArrayList();
		Iterator it = arrayList.iterator();
		while(it.hasNext()) {
		    …
		}