Intro to Computer Graphics: Just enough C++

Updated: September 10th, 2019

Slides by: Philmo Gu

Java vs C++

#1. Memory Management

C++

Managed
by developers using pointers. Supports structures and union.

Java

Controlled
by system, does not use pointers. Supports Threads and Interfaces.

#3. Runtime error detection mechanism

C++

Programmer's responsibility.

Java

System's
responsibility.

#2. Inheritance

C++

Provide
single and multiple inheritance both.

Java

Does
not support multiple inheritance. Uses the concept if Interface to achieve.

#4. Libraries

C++

Comparatively
available with low-level functionalities.

Java

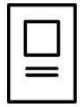
Provide
wide range of classes for various high-level services.

Source: https://www.educba.com/c-plus-plus-vs-java/

Java vs C++

#5. Program Handling

C++



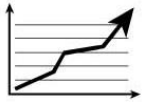
Methods
and data can reside outside classes.
Concept of global file, namespace
scopes available.

Java

All
methods and data reside in class itself.
Concept od Package is used.

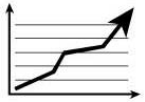
#6. Type Semantics

C++



Supports
consistent support between primitive and
object types.

Java



Different
for primitive and object types.

#7. Portability

C++

Platform
dependent as source code must be
recompiled for different platform.

Java

Uses
concept of bytecode which is
platform independent and can be
used with platform
specific JVM.

#8. Polymorphism

C++

Explicit
for methods, supports mixed hierarchies.

Java

Automatic,
uses static and dynamic binding.

Source: https://www.educba.com/c-plus-plus-vs-java/

Hello World!

For C++

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, world!";
    return 0;
}
```

For Java

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Preprocessors

- Separate programs that manipulate the text in each code file

```
1  #include <iostream>
2
3  #define MY_NAME "Alex"
4
5  int main()
6  {
7      std::cout << "My name is: " << MY_NAME;
8
9      return 0;
10 }
```

```
1  // The contents of iostream are inserted here
2
3  int main()
4  {
5      std::cout << "My name is: " << "Alex";
6
7      return 0;
8  }
```

Namespace

- An area code for identifier to be unique

```cpp
namespace Foo
{
    // This doSomething() belongs to namespace Foo
    int doSomething(int x, int y)
    {
        return x + y;
    }
}

namespace Goo
{
    // This doSomething() belongs to namespace Goo
    int doSomething(int x, int y)
    {
        return x - y;
    }
}

int main()
{
    std::cout << Foo::doSomething(4, 3) << '\n';
    std::cout << Goo::doSomething(4, 3) << '\n';
    return 0;
}
```

Source: https://www.learncpp.com/cpp-tutorial/4-3b-namespaces/

Namespace

- An area code for identifier to be unique

```
namespace Foo
{
    // This doSomething() belongs to namespace Foo
    int doSomething(int x, int y)
    {
        return x + y;
    }
}
```

```
namespace Goo
{
    // This doSomething() belongs to namespace Goo
    int doSomething(int x, int y)
    {
        return x - y;
    }
}
```

```
int main()
{
    std::cout << Foo::doSomething(4, 3) << '\n';
    std::cout << Goo::doSomething(4, 3) << '\n';
    return 0;
}
```

```
7
1
```

Source: https://www.learncpp.com/cpp-tutorial/4-3b-namespaces/

Overloading Operators

- **Operation** = mathematical calculation involving one or more inputs that produces a new value (output)
  - Operator = symbol(s) that specify an operation

```
Vec2f operator+(Vec2f a, Vec2f b) { return Vec2f(a.x + b.x, a.y + b.y); }
```

Source: https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/

Struct and Classes

- By Default:
  - Struct = *Public members*
  - Class = *Private members*

```
struct Vec2f {
    //public members by default
    Vec2f() = default;
    Vec2f(float x, float y) : x(x), y(y) {}

    float x = 0.f;
    float y = 0.f;
};
```

```
class Triangle {
    //Private members by default
    Vec2f m_verts[3];

public:
    Triangle() = default;
    Triangle(Vec2f a, Vec2f b, Vec2f c) {
        m_verts[0] = a;
        m_verts[1] = b;
        m_verts[2] = c;
    }
};
```

Values and Pointers

- Pointer = memory address to a value
  - & = address-of operator
  - * = dereference operator

```
1  int value = 5;
2  std::cout << &value; // prints address of value
3  std::cout << value; // prints contents of value
4
5  int *ptr = &value; // ptr points to value
6  std::cout << ptr; // prints address held in ptr, which is &value
7  std::cout << *ptr; // dereference ptr (get the value that ptr is pointing to)
```

Source: https://www.learncpp.com/cpp-tutorial/67-introduction-to-pointers/

Values and Pointers

- Pointer = memory address to a value
  - & = address-of operator
  - * = dereference operator

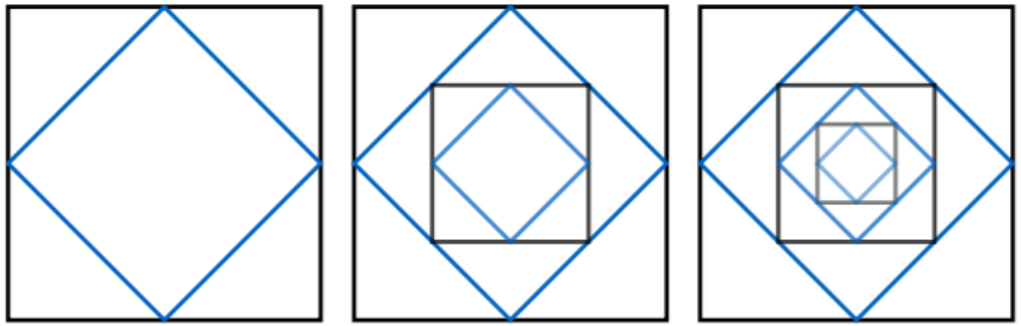
```
1  int value = 5;
2  std::cout << &value;     0012FF7C
3  std::cout << value; /    5
4
5  int *ptr = &value; /
6  std::cout << ptr; //     0012FF7C
7  std::cout << *ptr; /     5
```

Source: https://www.learncpp.com/cpp-tutorial/67-introduction-to-pointers/

Example Code

Exercises

- Create a pattern of nested squares and diamonds, and print out the vertices of each level
- Create the Sierpinski triangle using triangles, and print out the vertices of each iteration



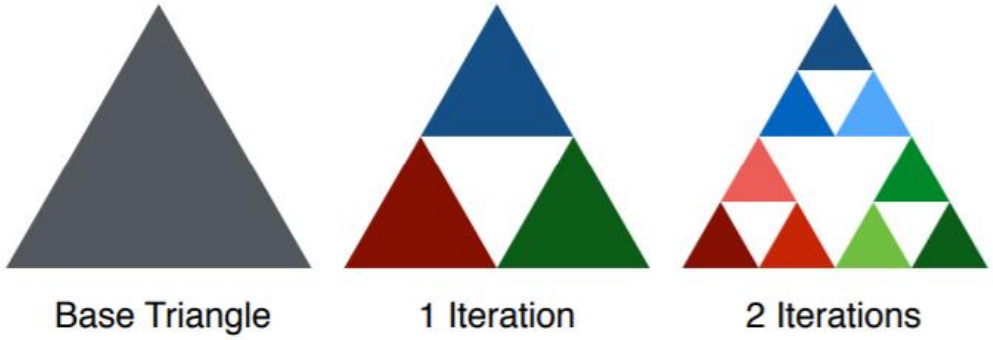


Image Credit: CPSC 453 (Fall 2018) Assignment 1*: Points, Lines and Triangles* by Sonny Chan