C++ crash course
Index:

1. Short introduction to C++
2. C++ VS Java
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
I'm sorry?!
So, C++ eh?
What's it all about?
I know nothing besides the name!

C++ is:
- Mostly based on C89/C99, and has C Syntax
- A language that supports object oriented design
- one of the most widespread programming languages ever
- Serious business
Wait! C-like Syntax and object oriented? Sounds just like JAVA! I know Java!

Well, yeah; It's pretty much just like Java

Well, except ... It's not at all!

Let me explain!
Index:

1. Short introduction to C++

2. C++ VS Java
   a) General structure
   b) Data types and Variables
   c) Classes and Objects
   d) Pointers and memory management
   e) Inheritance
   f) Vectors

3. Nasty C++ errors for everyone

4. How to create a documentation

5. Drawing with DirectX9
C++ VS Java

- focused on execution speed
- memory access at will
- lots of freedom for programmers
- allows operator overload
- compiled language
- no garbage collection by default
- complicated

- focused on programming speed
- memory access only through objects
- can restrict the programmer
- operators set in stone
- interpreted language
- built in garbage collection
- 'easy'
Index:

1. Short introduction to C++
2. C++ VS Java
   a) General structure
   b) Data types and Variables
   c) Classes and Objects
   d) Pointers and memory management
   e) Inheritance
   f) Vectors
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
Header *.h

- Class, member functions and variable declarations
- Defines public and private sections
- Virtual (abstract) function declaration
- Includes other header files and links libraries
- Acts as a kind of interface

Source *.cpp

- Constructor and destructor implementation
- All the actual code of member functions
- Member variable values are assigned
- Includes the corresponding header file
#pragma once
#include "Abs_Model.h"
#include "World.h"

class Game :
{
    public Abs_Model
{
    public:
        Game();
        ~Game();
        int getLandscapeWidth();
        int getLandscapeHeight();
        Fieldtype_enum::Fieldtype getHexfieldType(int x, int y);
    private:
        World* world_p;
};
#include "Game.h"

Game::Game()
{
    world_p = new World();
}

Game::~Game()
{
}

int Game::getLandscapeWidth()
{
    return world_p->getLandscapeWidth();
}

int Game::getLandscapeHeight()
{
    return world_p->getLandscapeHeight();
}

Fieldtype_enum::Fieldtype Game::getHexfieldType(int x, int y)
{
    return world_p->getHexfieldType(x, y);
}
Stop! Hey, Wep, what does “#pragma once” mean? And why are there “::” all over the place???

Oh? Well, “#pragma once” helps the compiler with this situation:

//File "grandparent.h"
struct foo
{
    int member;
};

//File "parent.h"
#include "grandparent.h"

//File "child.c"
#include "grandparent.h"
#include "parent.h"

Without “#pragma once” the compiler includes foo two times, and thus crashes, because of two declarations sharing the same name.
The “::” on the other hand tells a function or variable the namespace of which class it belongs to. Have a look at this modified code from “Game.cpp”

```cpp
#include "Game.h"

Game::Game()
{
    world_p = new World();
}

Game::~Game()
{
}

int getLandscapeWidth()
{
    return 0;
}

int Game::getLandscapeWidth()
{
    return world_p->getLandscapeWidth();
}
```

See, now there are 2 functions with the same name and type. The first is a global one, the second is the one specified in Game's class body.

Oh, did I mention there can be global functions and variables, that don't belong to any class, in C++? That's not possible in Java.
Speaking of global functions:
The main function always is a global function in C++

It's called “Main” for console applications and “WinMain” for window applications, but doesn't belong to any class' namespace. There is nothing like “Main::” in front of it!

Take note: Main always should be of the return type int!

Have a look at “Main.cpp”:

```cpp
#include "Main.h"

int WINAPI WinMain( HINSTANCE hinstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    static Abs_Model* game_p = new Game();
    static Abs_Controller* gamecontroller_p = new GameController( game_p);
    static Abs_View* userInterface_p = new UserInterface(hinstance,game_p, gamecontroller_p);

    return 0;
}
```
Index:

1. Short introduction to C++
2. C++ VS Java
   a) General structure
   b) Data types and Variables
   c) Classes and Objects
   d) Pointers and memory management
   e) Inheritance
   f) Vectors
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
Most simpler data types in C++ are just like in Java, however, note these differences:

1. Boolean type is called `bool` in C++

2. Strings are part of the `std` library, so you need to tell the namespace;
   An example:
   ```cpp
   std::string teststring = "abc";
   ```

3. C++ strings store **ASCII** characters, not **Unicode** characters as in Java

4. For lexicographic comparison on strings, the relational operators are used:
   ```cpp
   == != < <= > >=
   ```
   No need to use “equals” and “compareTo” as in Java.

5. “final” in Java is “const” in C++

6. C++ **does not check if local variables are initialized** before they are being used.
   The value of the variable is then the random bit pattern that happened to be in the variable's memory location.
Index:

1. Short introduction to C++
2. C++ VS Java
   a) General structure
   b) Data types and Variables
   c) Classes and Objects
   d) Pointers and memory management
   e) Inheritance and Interfaces
   f) Vectors
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
There are some major differences between classes and objects in Java and C++:

1. As stated, class definition only contains the **declarations** of the methods. The actual **implementations** are listed separately.

2. The most prominent difference is the behavior of object variables. In C++, object variables hold **values**, not object **references**:
   ```
   World example_world = World();
   This is very different from Java. There, this command would create and save a reference to the World.
   In C++, the World object itself is stored in that variable.
   ```

3. Following from the above:
   ```
   World another_world = example_world;
   Will copy the actual World object in C++, and not just a reference!
   ```

4. Problem:
   Two object variables cannot access the same object. If you need this in C++, then you need to use....
Index:

1. Short introduction to C++
2. C++ VS Java
   a) General structure
   b) Data types and Variables
   c) Classes and Objects
   d) Pointers and memory management
   e) Inheritance and Interfaces
   f) Vectors
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
Pointers in C++ have to be explicitly declared as such:

```cpp
static Abs_Model* game_p = new Game();
```

You can reference the object now by using:

```cpp
(*game_p).doSomethingSupercool()
```

But that's just ugly,

so the C++ guys also came up with this, doing the same:

```cpp
game_p->doSomethingSupercool()
```

And if you want to create a pointer to an object you already have use this:

```cpp
Abs_Model* game_p = &someGameObject;
```

& in general returns the address in memory of the thing that follows
That's not too hard to get, pretty easy actually!

Well, remember C++ doesn't have a garbage collector!

Not a problem when you create an object directly on the stack:

```cpp
World example_world = World();
```

It gets deleted once scope ends; **but**:

```cpp
World* example_world = new World();
```

That one doesn't, as it's on the heap! **MEMORY LEAK**!

You need to call its destructor once no longer needed:

```cpp
delete example_world;
```
Well, as long as you remember to delete all you objects you should be fine, don't you?

Only if you can be sure there's no error thrown before the object is deleted (read: never)!

```cpp
World* example_world = new World();
.
.
Error is thrown here!
.
.
dele te example_world;
```

There you have it: MEMORY LEAK!!

*Sigh* stupid Pointers!

Yea, fortunately, there's also **Smart** Pointers!
Smart pointers are a container that take Pointers.

Once the smart pointer goes out of scope, it automatically calls the destructor on the object it is pointing at.

There are 3 basic archetypes at this time in C++:

1. unique_ptr:
   This smart pointer cannot be copied. It can only be moved from one variable to the other.

2. shared_ptr:
   Reference-counted pointer, memory will only be freed if all instances of the shared_ptr in the program are gone.
   **Problem:** cyclical references!

3. weak_ptr:
   Like shared_ptr, but not reference-counted.
   => Object deleted if only weak_ptr point at it.
An example for the use of `std::unique_ptr`:

**File: DirectXRenderer.h:**

    std::unique_ptr<Vertex[]> landscapeVerticesArray_unqp;

**File: DirectXRenderer.cpp:**

    landscapeVerticesArray_unqp.reset(new Vertex [lw*lh*6]);
    [...]
    landscapeVerticesArray_unqp[x*lh * 6 + y * 6 + hexVertexNr] =
    { (FLOAT)(x)*hexWidthInPixel, (y + 0.25f)*hexHeightInPixel, 1.0f, 1.0f, fieldColor };
Well, there is one freely available for C and C++ as well in the form of an optional package!

It's called the **Boehm–Demers–Weiser garbage collector** and uses a mark-sweep algorithm.

The neat thing is: It even works with most **unmodified** programs, because it simply overwrites the default ways to allocate memory, so no changes to the code are necessary.

Can also be used for leak detection, pretty neat thing!
Index:

1. Short introduction to C++
2. C++ VS Java
   a) General structure
   b) Data types and Variables
   c) Classes and Objects
   d) Pointers and memory management
   e) Inheritance and Interfaces
   f) Vectors
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
C++ supports multiple inheritance, but no Interfaces. They aren't needed anyway, the only reason those exist in Java is the lack of multiple inheritance. Here is an example for an abstract class in C++:

```cpp
#pragma once
#include "Fieldtype_enum.h"

class Abs_Model
{
public:
    virtual ~Abs_Model();
    virtual int getLandscapeWidth() = 0;
    virtual int getLandscapeHeight() = 0;
    virtual Fieldtype_enum::Fieldtype getHexfieldType(int x, int y) = 0;
};
```

There's no constructor (guess why), but a **virtual destructor is needed** so the destructor of the child class is called once an object referenced by a pointer to `Abs_Model` is deleted. `virtual Function() = 0;` means the function is 100% abstract and must be implemented by the child class.
Let's have a look at the child class again!
You've seen it before.

This is the syntax to inherit from another class:
```
#include "Abs_Model.h"
#include "World.h"
class Game : 
  public Abs_Model
{
  public:
    Game();
    ~Game();
    int getLandscapeWidth();
    int getLandscapeHeight();
    Fieldtype_enum::Fieldtype getHexfieldType(int x, int y);
  private:
    World* world_p;
};
```

Note: It is important to use `public` inheritance. Otherwise, the functions below would not overwrite the virtual functions we did see before, default inheritance is `private`.
Index:

1. Short introduction to C++
2. C++ VS Java
   a) General structure
   b) Data types and Variables
   c) Classes and Objects
   d) Pointers and memory management
   e) Inheritance and Interfaces
   f) Vectors
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
The C++ vector construct combines the best features of arrays and vectors in Java, it is fast, but can grow dynamically.

```cpp
vector<int> example_v;
creates an initially empty vector.

vector<int> example_v(10);
creates a vector that initially has 10 elements.

You can add more elements with the push_back method, or resize the vector:
example_v.push_back(n);
example_v.resize(width);
```

element_v.pop_back() removes the last element from example_v.
Use the size method to find the current number of elements in example_v.

You access the elements with the familiar [] operator.
sum = sum + example_v[i];

```cpp
vector<int> b = a;
This copies all elements from a to b; Their sizes better match, because C++ doesn't check it!
```
Index:

1. Short introduction to C++
2. C++ VS Java
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
What happens if two files need to include each other, and you do exactly this in both header files? Compiler Errors arise!
Here is how to do it properly:

File: MainWindow.h:

```cpp
#pragma once
#include <windows.h>
#include "Abs_Model.h"

//forward declaration to be overwritten later
class DirectXRenderer;

 [...] 
/!
Used to render the actual images that are being displayed inside the window
*/
DirectXRenderer* DirectXRenderer_p;
```

File: MainWindow.cpp:

```cpp
// overwrites the forward declaration of the DirectXRenderer class in MainWindow.h,
// as circular inclusions among header files are not allowed
#include "DirectXRenderer.h"
```

In Contrast, the DirectXRenderer class can include in the normal way!
If a switch statement appears to always be choosing the last option, you might have forgotten \texttt{break}!

```cpp
switch (randomNumber) {
  case 0:
    hexgrid_v_unqp[x][y].reset(new GrasslandField());
    // never EVER forget \texttt{break}; else the switch executes the code in all
    // remaining cases as well, regardless of the \texttt{variable} matching the case
    break;
  case 1:
    hexgrid_v_unqp[x][y].reset(new HillsField());
    break;
  case 2:
    hexgrid_v_unqp[x][y].reset(new ForestField());
    break;
  case 3:
    hexgrid_v_unqp[x][y].reset(new MountainsField());
    break;
  case 4:
    hexgrid_v_unqp[x][y].reset(new WastelandField());
    break;
  case 5:
    hexgrid_v_unqp[x][y].reset(new OceanField());
    break;
}
```
Index:

1. Short introduction to C++
2. C++ VS Java
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
Comments to be used with Doxygen need to use this kind of syntax:

```c
/*!
Registers a new window class to the OS and assigns values to instance variables inherited from the WNDCLASS struct

\param hinstance The unique instance handle we received from the operating system into our first entry point parameter when our program started, will be bound here into the window class data structure.

\param windowClassname The human readable name of the type of window (in the sense of a visible object e.g. a button or a textfield) we are about to create. This name works as a reference when telling windows what kind 'style' of a window we want to create
*/

void RegisterWindowClassToOS(HINSTANCE hinstance, LPCWSTR windowClassname);
```
Index:

1. Short introduction to C++
2. C++ VS Java
3. Nasty C++ errors for everyone
4. How to create a documentation
5. Drawing with DirectX9
Swapping Pointers between front and back buffer to avoid tearing:

The Image being displayed now!

The next Image to be displayed!
Multiple Back Buffers Gets Better Performance

Let's say that every once in a while you finish rendering the back buffer and are ready to swap, but the screen hasn't yet finished drawing the contents of the front buffer. Swapping now would cause more tearing. So what happens instead is your program stops and waits for the screen to finish. You could, of course, be spending this valuable time preparing the next image, and having multiple back buffers allows your program to do just that.