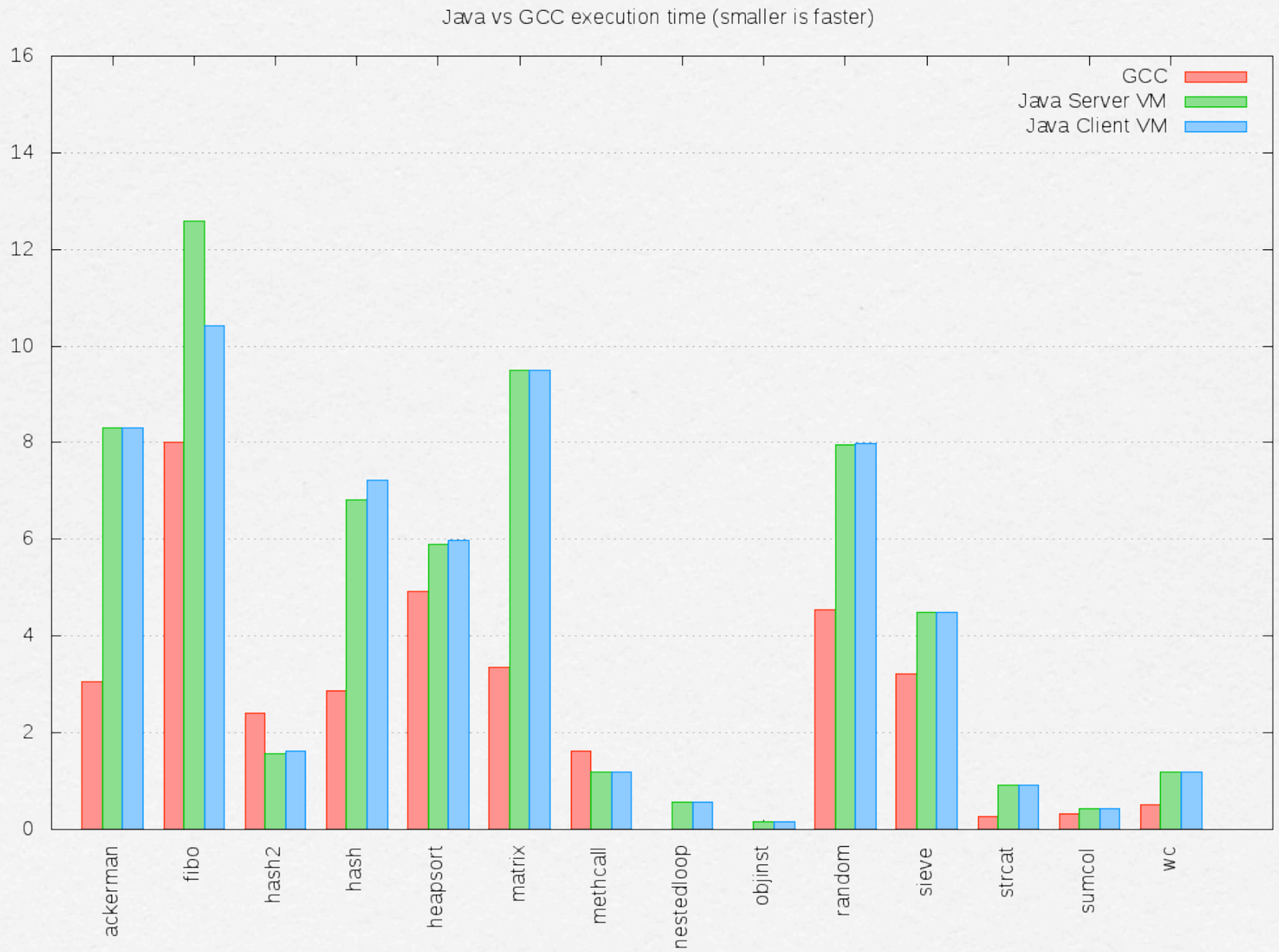
C++ Basics

By Max Lam

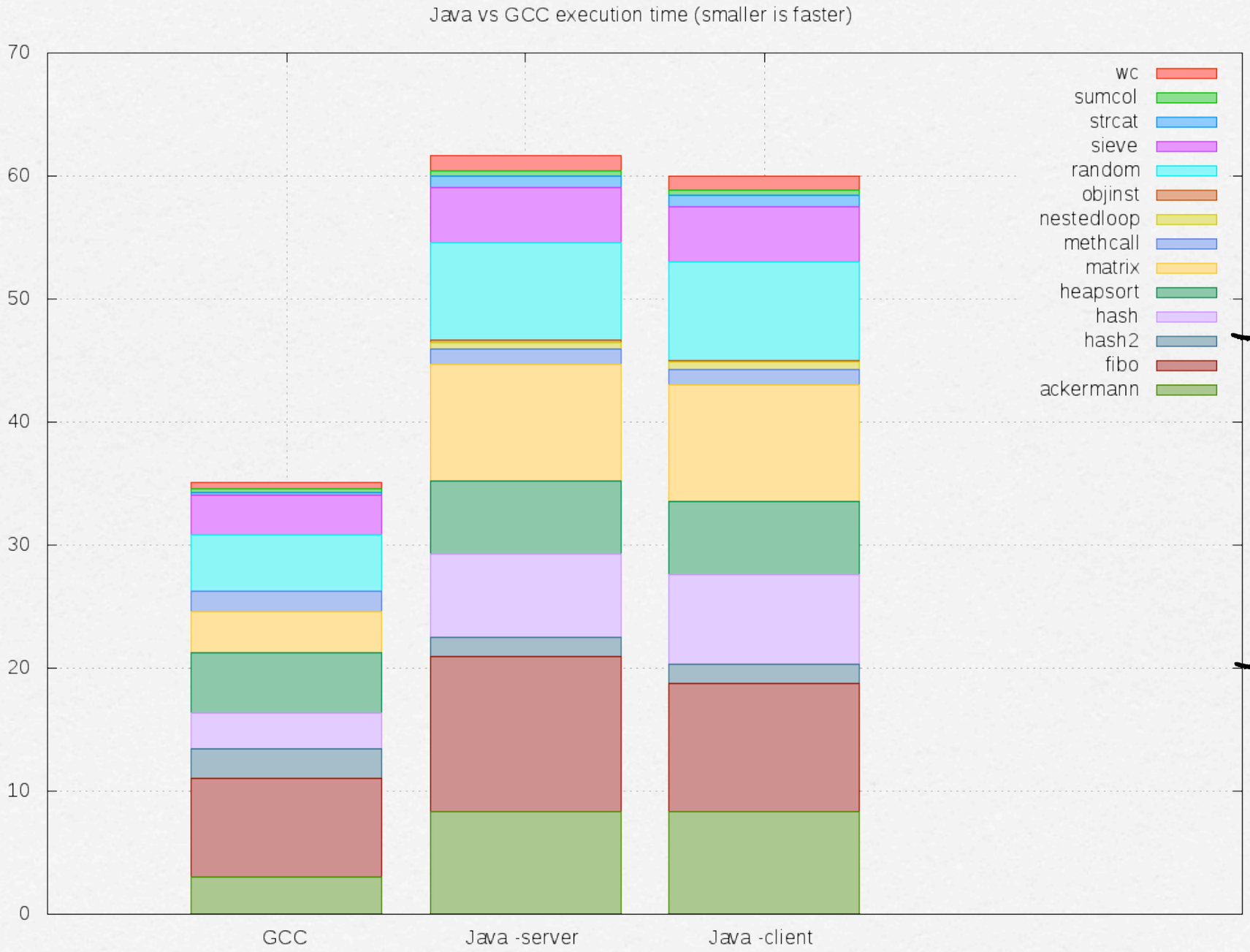
Why C++

- It's really fast
- The roadrunner of languages

How fast? (Speed Comparison)



How fast? (Cumulative Comparison)



C++ time:

35 sec

Java Server VM time:

61.65 sec

Java Client VM time:

60.05 sec

That's pretty fast.

Nearly 2x faster than java in some cases

Why C++

- It's a totally different beast

Why C++

- It's a totally different beast

Java

Why C++

- It's a totally different beast

Java

vs

C++

How different of a Beast?

- c++ was designed for systems and applications programming
- c++ was designed to be efficient
- Java was designed initially for printing systems and network computing
- Java was designed to be portable

Why C++

- It's humongous
- Used in awesome companies

C++ is pretty awesome.

- To sum it up:
  - C++ is really fast
  - C++ has a different purpose
  - C++ is used by many awesome companies

Caveat

- Many People dread the intensity of C++ memory management
- C++ memory management can get funky
- However, by letting the user manage memory (unlike in java) speed increases significantly
- This major setback is actually an advantage

Therefore:

- c++ is not for the faint of heart

Let's get started

The ‘Main’ Function

Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println(“Hello world\n”);
    }
}
```

C++

```
#include <iostream>

int main(int argc, char *argv[]) {
    cout << “Hello world!” << endl;
}
```

* If you don’t want to pass arguments, do ‘int main(void) {‘ instead

Arguments in Java
and
Arguments in C++

Don’t worry if this looks really weird. We will learn more about arguments in later slides. (char *argv[] is actually a really primitive array; a vestige from the original C language)

The ‘Main’ Function

Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println(“Hello world\n”);
    }
}
```

Printing in Java
and
Printing in C++

C++

```
#include <iostream>

int main(int argc, char *argv[]) {
    cout << “Hello world!” << endl;
}
```

However, the ‘cout’
function requires

The ‘cout’ function

- It prints stuff
- You absolutely need to have: ‘include <iostream>’
- You can chain strings together
  - cout << “Hello, ” << “World!” << endl;
  - cout << “Hello, World!” << endl;
  - cout << “Hi” << “there” << “my” << “name” << “is” << “Bob!”
- This can be pretty useful
- Also, “endl” is the same as a newline or “\n”
- You can also print out various variables such as strings, chars, ints, etc... we will get to that next

Ancient Types

These variable types are from the original C language. These include

int
char
bool

declaring variables

assigning variables values

```
#include <iostream>

int main(int argc, char *argv[]) {
    int a, b, c;
    char d, e, f;
    bool awesome;

    a = 1;
    b = 2;
    c = 3;

    d = ‘x’;
    e = ‘y’;
    f = ‘z’;

    awesome = true;
}
```

* Beware that ‘x’ is not the same as “x”

Ancient Types

These variable types are from the original C language. These include

int
char
bool

You can do the general variable evaluations as in Java

You can also do basic integer addition, subtraction, etc... as in Java

```
#include <iostream>

int main(int argc, char *argv[]) {
    int a, b, c;
    bool awesome;

    int a = 1;
    int b = 2;
    int c = 3;
    awesome = true;

    if ((a + b + c) >= 6 &&
        awesome == true) {
            cout << “awesome-ness!”;
    }
    a += 5;
    b++;
}
```

More Ancient Types

Primitive arrays:

```
int numbers[20];
char letters[50];
```

(They're like ArrayLists. Except harder to use. This will be resolved in later slides.)

Declaring arrays of different types

Assigning the values of each element of my arrays

More ways of declaring arrays of different types

Causing my program to crash

```
#include <iostream>

int main(int argc, char *argv[]) {
    int numbers[20];
    char letters[50];

    for (int i = 0; i < 20; i++) {
        numbers[i] = 5;
    }

    for (int i = 0; i < 50; i++) {
        letters[i] = 'j';
    }

    int moreNumbers[5] = {1, 2, 3, 4, 5};
    char moreLetters[3] = {'a', 'b', 'c'};

    numbers[200] = 5;
}
```

*** #s in between brackets specifies array size. You must have them!**

Things so far

- C++ is a lot like Java
  - There are basic comparison operators
  - Variables behave the same
  - Functions, which haven't been shown, are basically the same as well.
  - Primitive types include int, char and bool, which are really similar to Java's int, char and bool
  - Lot of the same features
- But, things will start to get different

A Curious Problem

What does this print?

```
#include <iostream>

/* Adds 2 numbers, a and b, and stores result
 * in a
 */
void add(int a, int b) {
    int a = a + b;
}

int main(int argc, char *argv[]) {
    int a = 5, b = 6;
    add(a, b);
    cout << a << endl;
}
```

A Curious Problem

What does this print?

```
#include <iostream>

/* Adds 2 numbers, a and b, and stores result
 * in a
 */
void add(int a, int b) {
    int a = a + b;
}

int main(int argc, char *argv[]) {
    int a = 5, b = 6;
    add(a, b);
    cout << a << endl;
}
```

Answer: 5

A Curious Problem

```
#include <iostream>

/* Adds 2 numbers, a and b, and stores result
 * in a
 */
void add(int a, int b) {
    int a = a + b;
}

int main(int argc, char *argv[]) {
    int a = 5, b = 6;
    add(a, b);
    cout << a << endl;
}
```

You'd think that the value of 'a' was changed in the function 'add'. However, this is false. 'a' is not changed at all -- in actuality, only a copy of 'a' was changed, not the actual 'a'

A Curious Problem

```
#include <iostream>

/* Adds 2 numbers, a and b, and stores result
 * in a
 */
void add(int a, int b) {
    int a = a + b;
}

int main(int argc, char *argv[]) {
    int a = 5, b = 6;
    add(a, b);
    cout << a << endl;
}
```

Every time parameters are passed to a function, local copies are made of these variables, and only these copies are changed in the function. All local copies of variables are destroyed at the end of a function

The Solution: Pointers

- Pointers are references to the addresses of a variable
- You can have a pointer to any type: int, char, bool ,etc...
- Notation for a pointer is the asterisk: *

The Solution: Pointers

```
#include <iostream>

int main(int argc, char *argv[]) {
    int * a;
    char * b;

    *a = 5;
    *b = ‘a’;

    b = 25;

    if (*b == ‘a’) {
        cout << “b is 5” << endl;
    }
    if (*a > 5) {
        cout << “a is greater than 5” << endl;
    }
}
```

Declaring pointers

Setting the value of the pointer

Setting the actual address of a pointer
(This is not good. Don’t do this)

Doing stuff with pointers (comparisons)

The Solution: Pointers

More w/ Pointers

All variables have addresses

& = address of operator

```
#include <iostream>

int main(int argc, char *argv[]) {
    int *a, myAwesomeNumber;
    myAwesomeNumber = 42;

    a = &myAwesomeNumber;

    if (*a == 42) {
        cout << “This number really is awesome; 42” << endl;
    }

    *a = 0;
}
```

Setting integer pointer ‘a’ to have the address of ‘myAwesomeNumber’

How would this affect myAwesomeNumber?

The Solution: Pointers

More w/ Pointers

All variables have addresses

& = address of operator

```
#include <iostream>

int main(int argc, char *argv[]) {
    int *a, myAwesomeNumber;
    myAwesomeNumber = 42;

    a = &myAwesomeNumber;

    if (*a == 42) {
        cout << “This number really is awesome; 42” << endl;
    }

    *a = 0;
}
```

Setting integer pointer ‘a’ to have the address of ‘myAwesomeNumber’

How would this affect myAwesomeNumber?

A: It would change ‘myAwesomeNumber’s value from 42 -> 0

The Solution: Pointers

Back to original Problem: Altering parameter value from a function

Passing a pointer will solve the problem

```
#include <iostream>

void add(int *a, int b) {
    *a = *a + b;
}

int main(int argc, char *argv[]) {
    int a = 5, b = 6;
    add(&a, b);
}
```

(Remember to use the variable's address!)

Why this works

- Although all parameters are only copies, copies of pointers still maintain the address of the original variable
- So when we seemingly alter the pointer, we are actually altering what's stored in its address, the original variable

C++ Easiness

This solution works too!

```
#include <iostream>

void add(int &a, int b) {
    a = a + b;
}

int main(int argc, char *argv[]) {
    int a = 5, b = 6;
    add(a, b);
}
```

Declaring a function to pass by reference
(Basically does the pointer stuff in the background)

The '&' sign here does not symbolize the 'address of' operator

Don't need to pass address of variable anymore!!

Just a note

```
#include <iostream>

void add(int &a, int b) {
    a = a + b;
}

int main(int argc, char *argv[]) {
    int a = 5, b = 6;
    add(a, b);
}
```

This is an array of pointers

However, pointers can also act as strings

So, this is just an array of strings

(Don’t worry, the rest of C++ isn’t that scary, Though the C is)

Standard Library Stuff

- Provides easy features to use
- It's like Java's built-in ArrayList, and Hashmap, etc...
- C++ provides (not by any means an exhaustive list)
  - Vectors ~ ArrayList
  - Maps ~ Hashmaps
  - String ~ Java string?

Vectors

```
#include <iostream>
#include <vector>

using namespace::std;

int main(int argc, char *argv[]) {
    vector<int> myNumbers;

    myNumbers.push_back(5);
    myNumbers.push_back(6);
    myNumbers.push_back(7);
    myNumbers.push_back(8);

    for (int i = 0; i < myNumbers.size(); i++) {
        cout << myNumbers[i] << endl;
    }
}
```

Need to include vector

Declaring a vector of type int

Adding numbers

Getting the number of elements inside the vector

Accessing individual elements of vector

Vectors

- So, it's a lot like using an ArrayList

Maps

- They are like Java HashMaps
- Use them the same way you use vectors
- Notations are basically the same

Creating Classes

- You can create your own classes in C++, like in Java!
- Things are a bit different

Creating Classes

```
#include <iostream>
#include <string>

class Car {
public:
    Car();
    ~Car();
private:
    int numWheels;
    string color;
}

//Constructor
Car::Car() {
    numWheels = 4;
    color = “black”;
}

//Destructor
Car::~Car() {

}
```

Constructor and Destructor

Privates!!
(inaccessible to the outside)

Called when class is created

Called when class is destroyed

I have an error here. What is is?

Creating Classes

```
#include <iostream>
#include <string>

class Car {
public:
    Car();
    ~Car();
private:
    int numWheels;
    string color;
};

//Constructor
Car::Car() {
    numWheels = 4;
    color = “black”;
}

//Destructor
Car::~Car() {

}
```

Constructor and Destructor

Privates!!
(inaccessible to the outside)

Called when class is created

Called when class is destroyed

I have an error here. What is is?

Creating Classes

```
#include <iostream>
#include <string>

class Car {
public:
    Car();
    ~Car();
private:
    int numWheels;
    string color;
};

//Constructor
Car::Car() {
    numWheels = 4;
    color = “black”;
}

//Destructor
Car::~Car() {

}
```

```
#include <iostream>
#include <string>
#include “Car.h”

//Car declared in Car.h

int main(int argc, char *argv[]) {
   Car newCar = Car();
}
```

Creates new class
and also calls its constructor

Creating Classes

```
#include <iostream>
#include <string>

class Car {
public:
    Car();
    ~Car();

    void startEngine();
private:
    int numWheels;
    string color;
};

//Pretend constructor
//and destructors are here

Car::startEngine() {
    cout << “Vroom Vroom!” << endl
}

}
```

```
#include <iostream>
#include <string>
#include “Car.h”

//Car declared in Car.h

int main(int argc, char *argv[]) {
    Car newCar = Car();

    newCar.startEngine();
}
```

Vroom Vroom!

Classes

- Are pretty versatile
- You can do a lot with them!
- This is only a skin-deep introduction to C++ classes
- Only a skin-deep introduction to C++ in general

End of Basics of C++