CS 241: Systems Programming
Lecture 8. Introduction to C

Spring 2020

Prof. Stephen Checkoway

Hello, World!

```c
#include <stdio.h>

int main(void) {
  printf("Hello world!\n");
  return 0;
}
```

Functions

```
/* Function declaration.
 * - No return value.
 * - Has three parameters, parameter names are optional.
 * - Ends with a semicolon.
 */
void foo(int x, float y, char z);

/* Function definition.
 * - Must match declaration.
 * - Parameter names are not optional.
 * - Body of function wrapped in { }.
 */
void foo(int x, float y, char z) {
  /* ... */
}
```

Main function

```
// The main function is where execution begins.
// - Returns an int, 0 is success, 1-127 are failure.
// - Takes 0, 2, or implementation-defined number of parameters.
// - argc is the number of command line parameters.
// - argv points to an array of command line parameters.
int main(void) { /* ... */ }
int main(int argc, char **argv) { /* ... */ } // Use this one.
int main(int argc, char **argv, char **envp) { /* ... */ }
```

Jobs of a Compiler

Inputs

- C program file and options
- Libraries

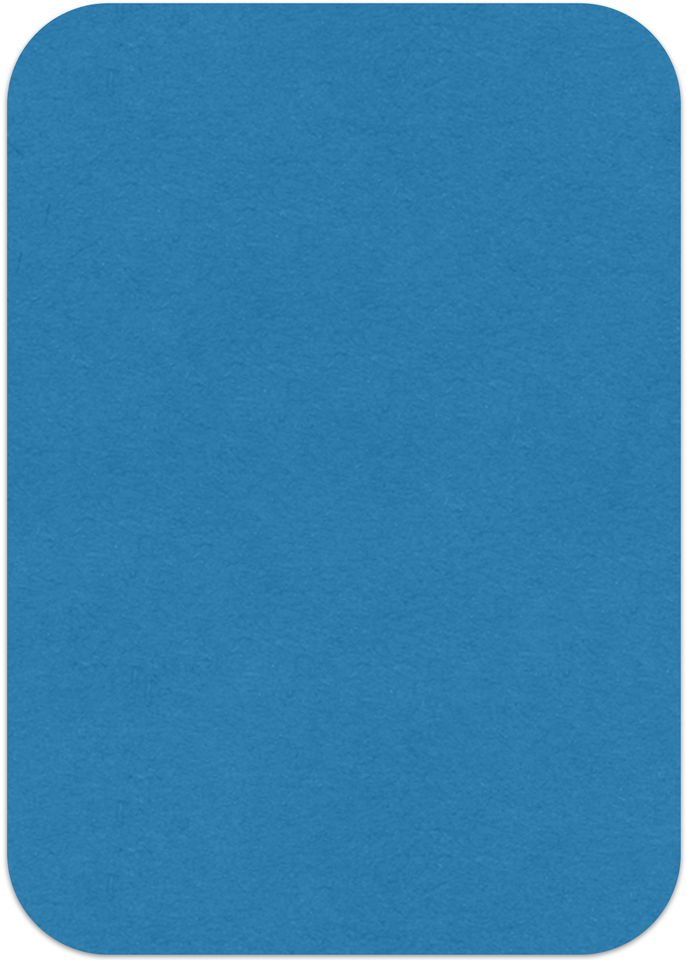
Compilation phases

- Preprocessing
- Compilation
- Linking

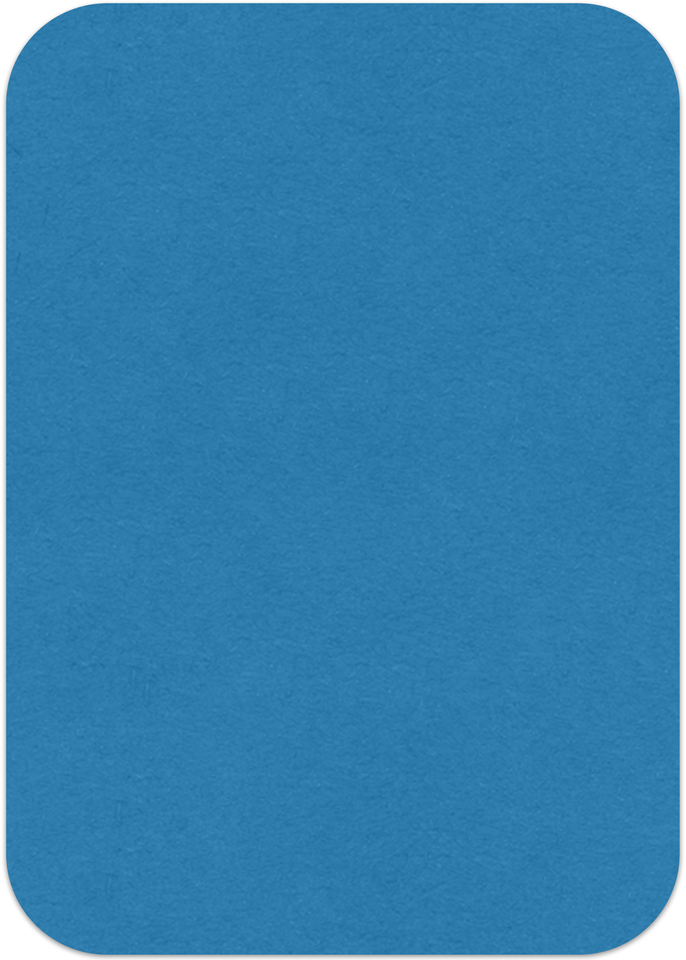
Outputs

- Executable
- Warnings and errors

Compilation

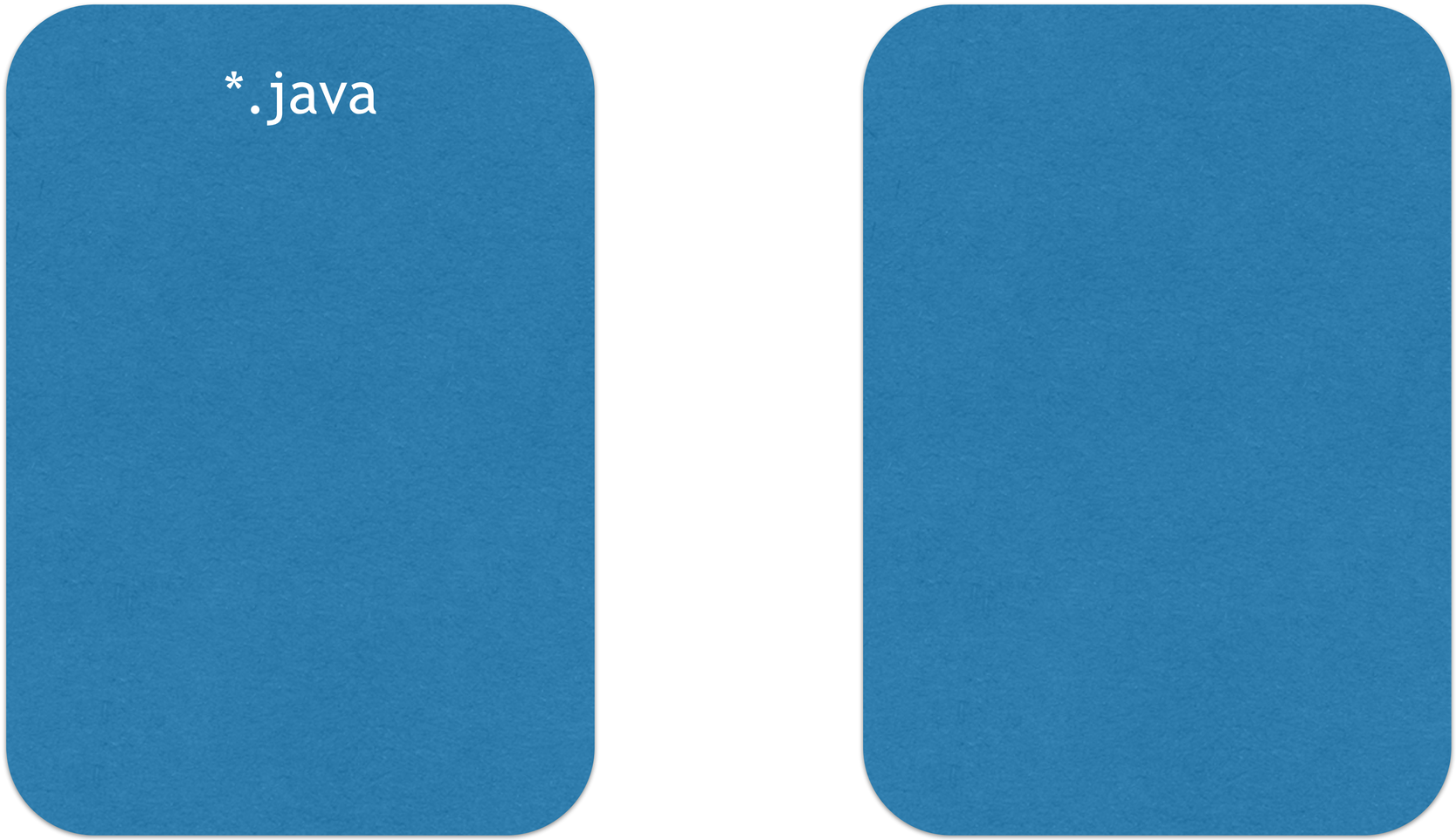


Java Model



C Model

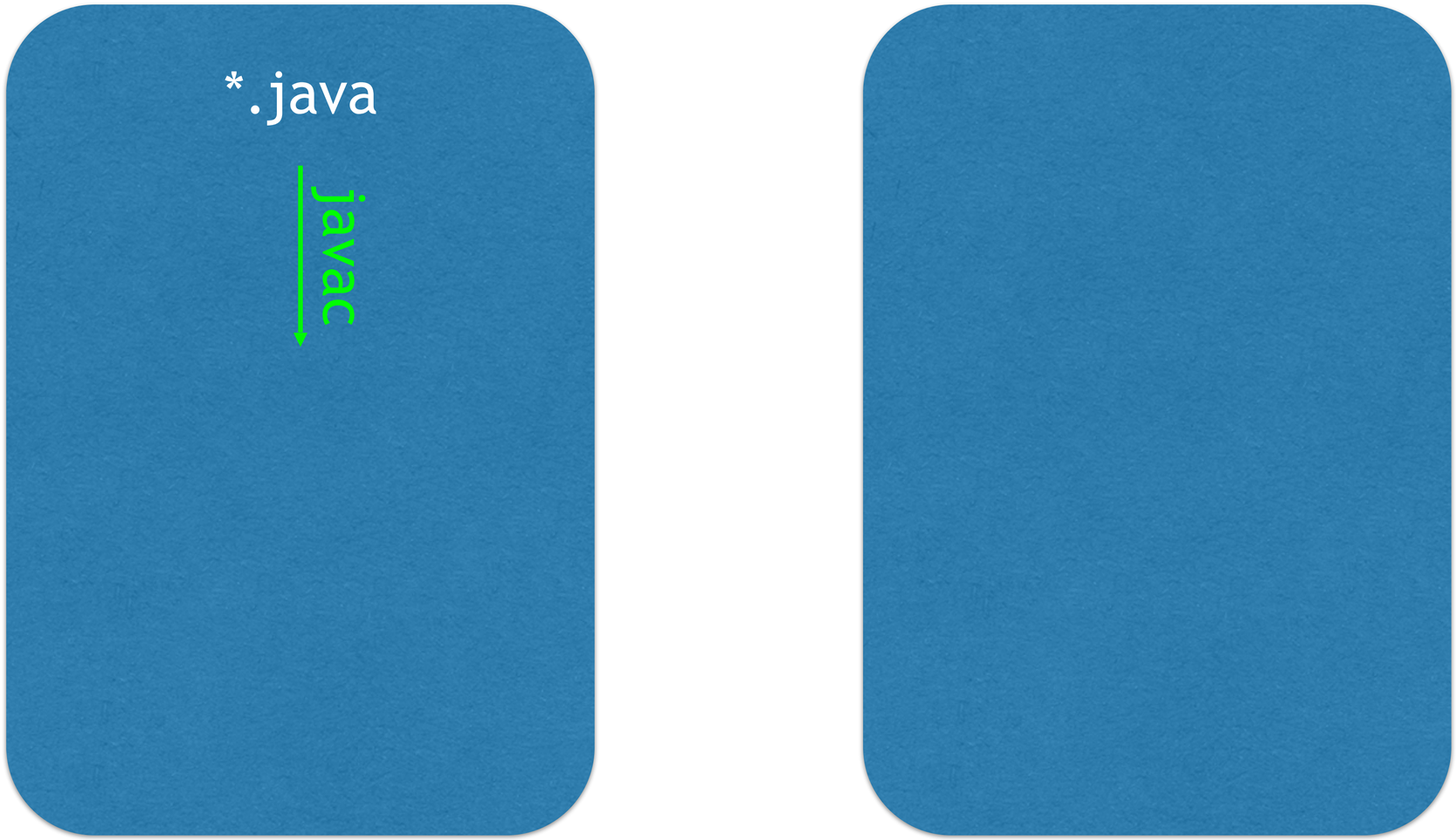
Compilation



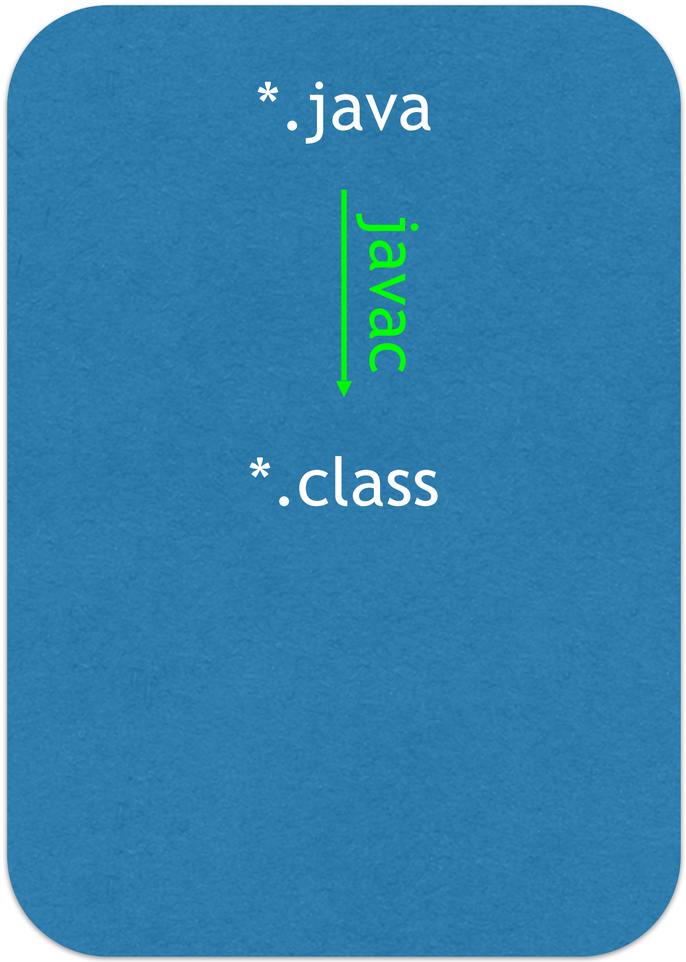
Java Model

C Model

Compilation



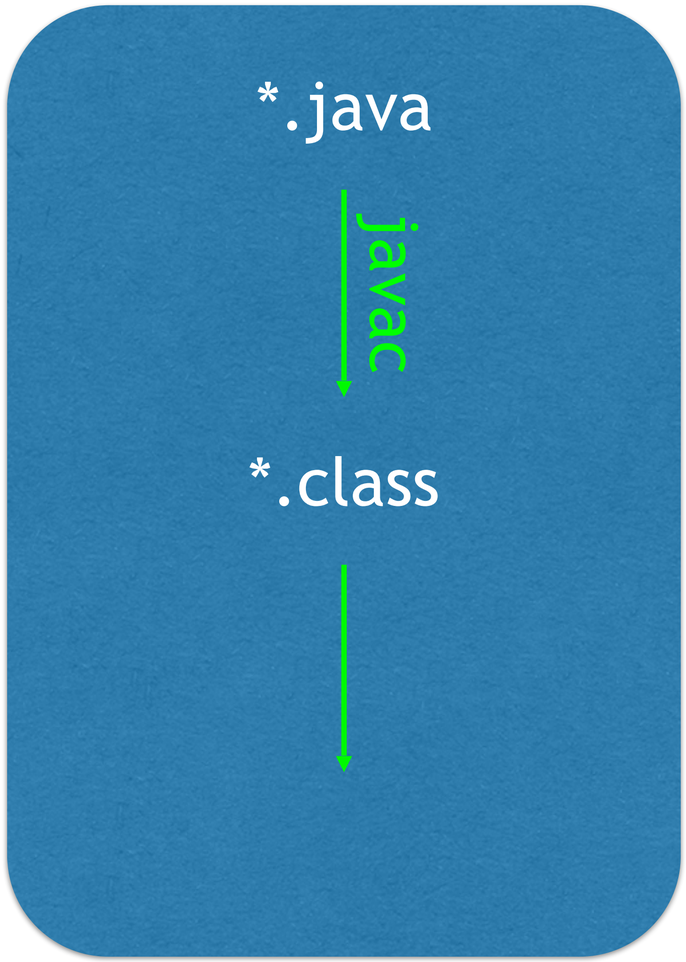
Compilation



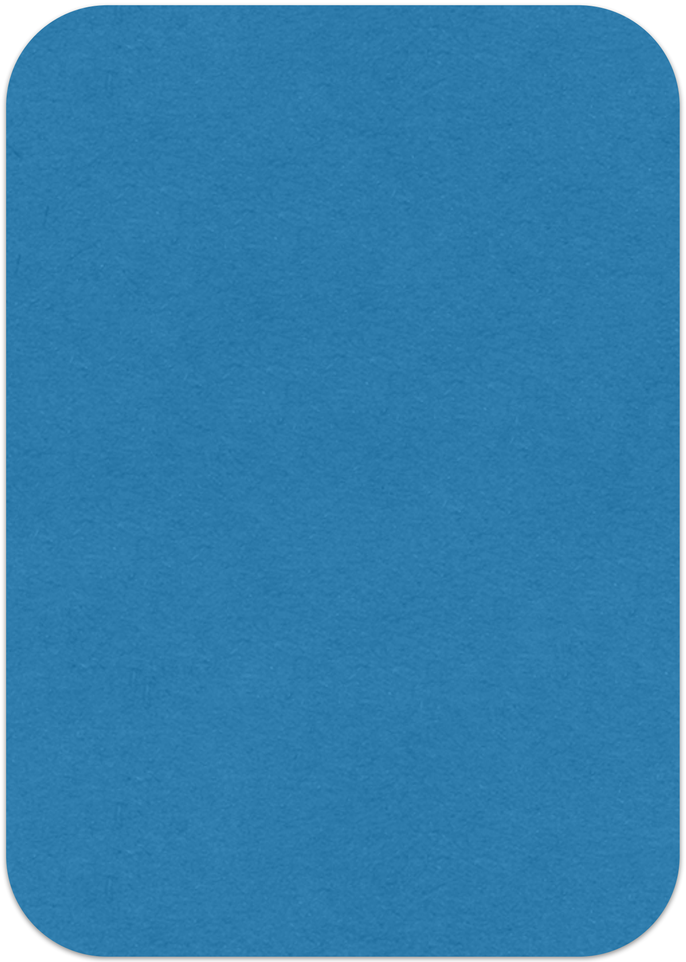
Java Model

C Model

Compilation

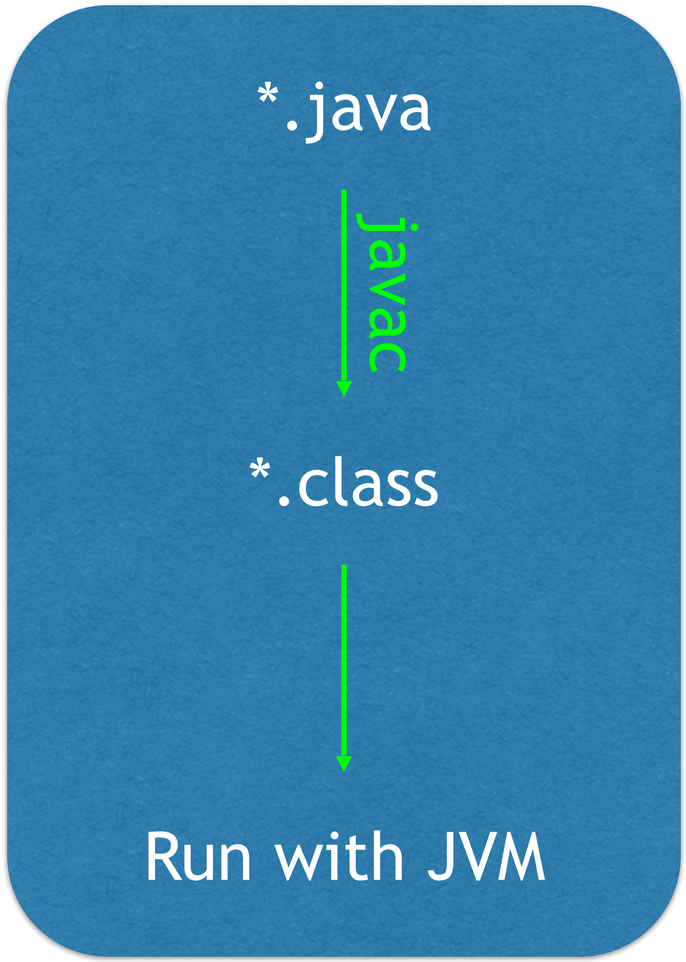


Java Model

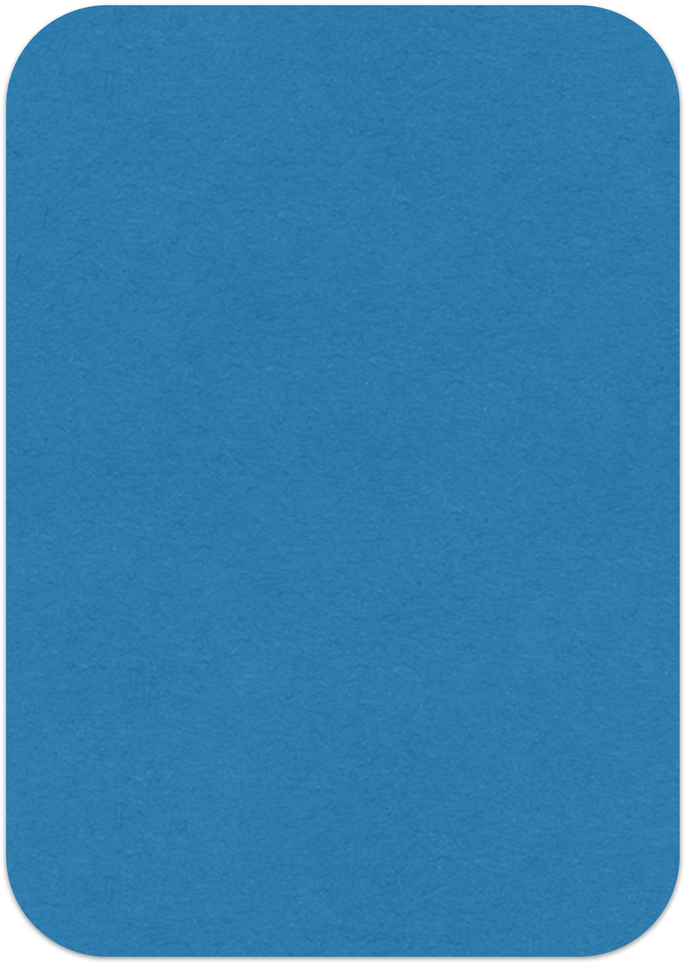


C Model

Compilation

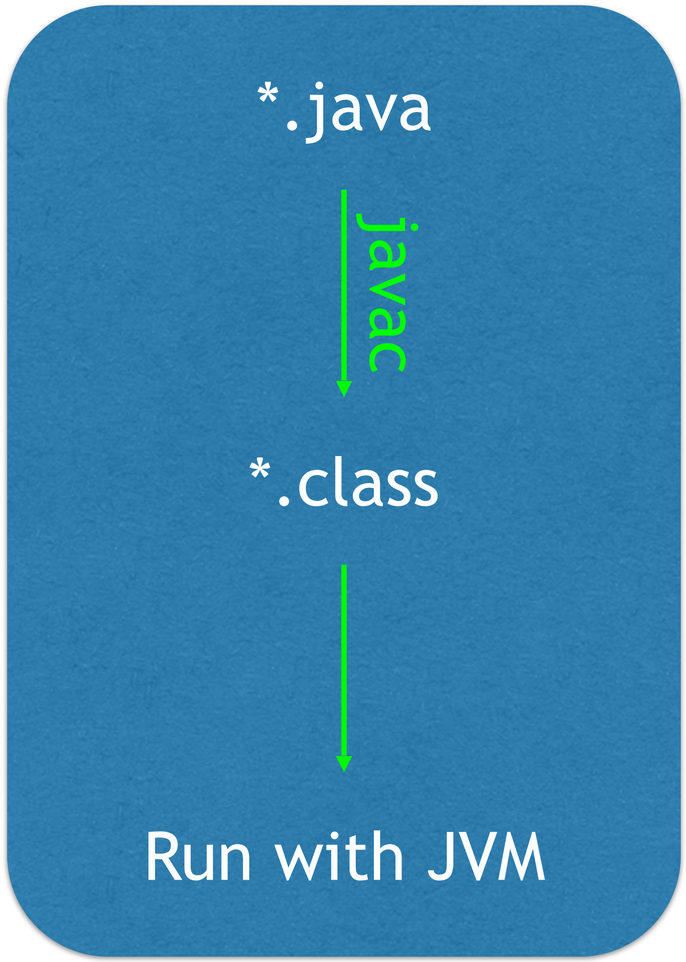


Java Model

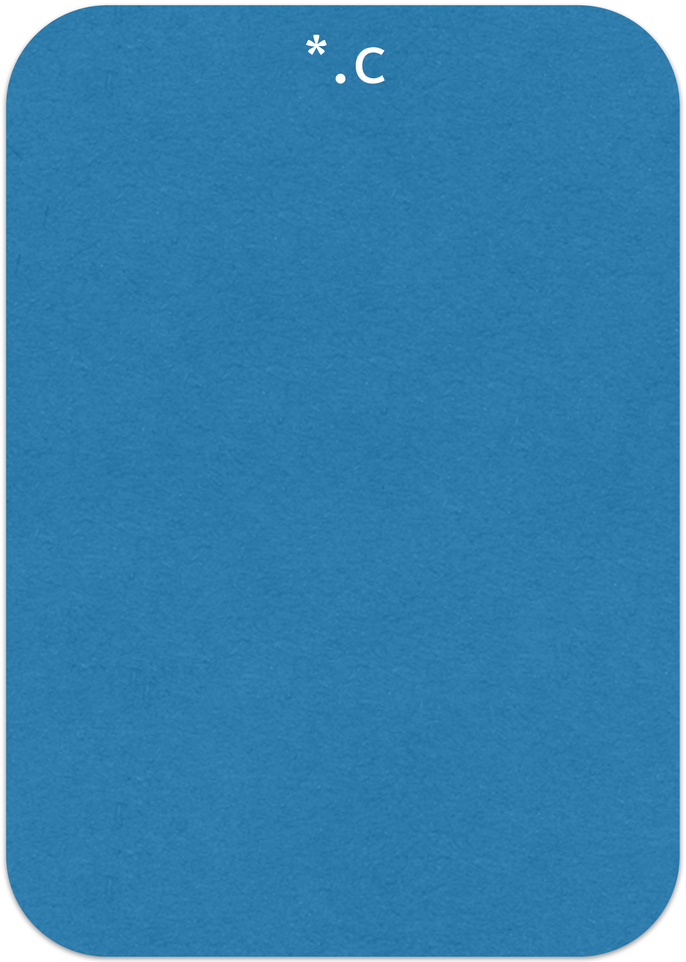


C Model

Compilation

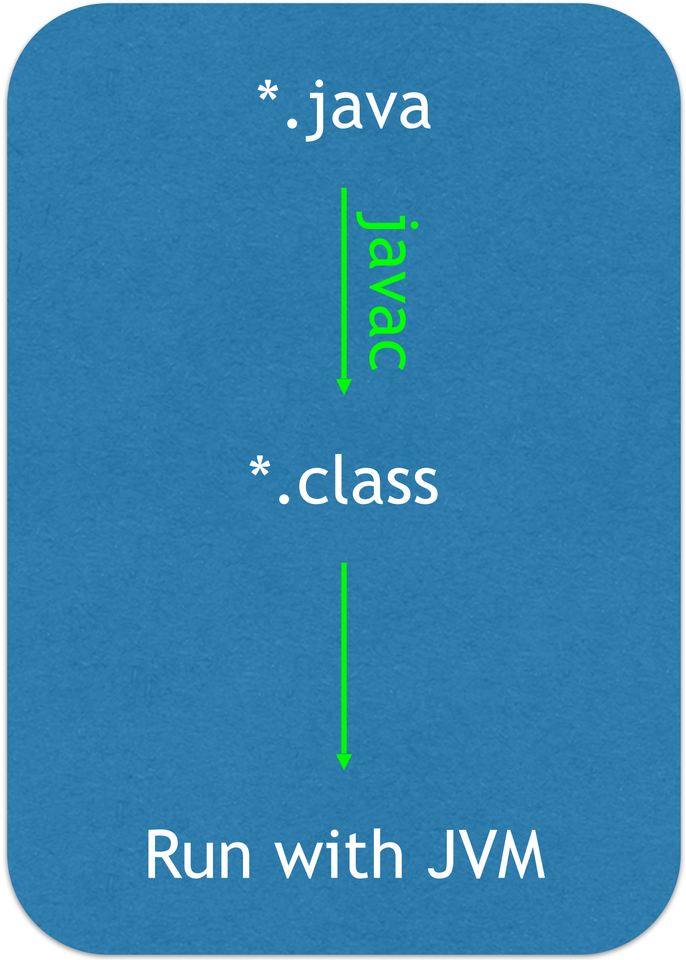


Java Model

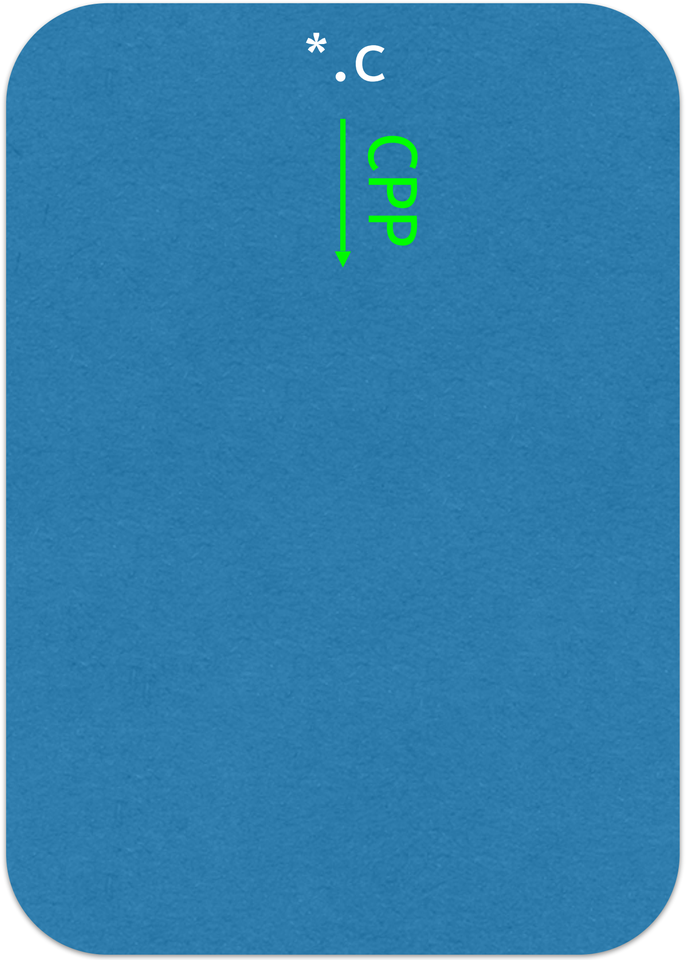


C Model

Compilation

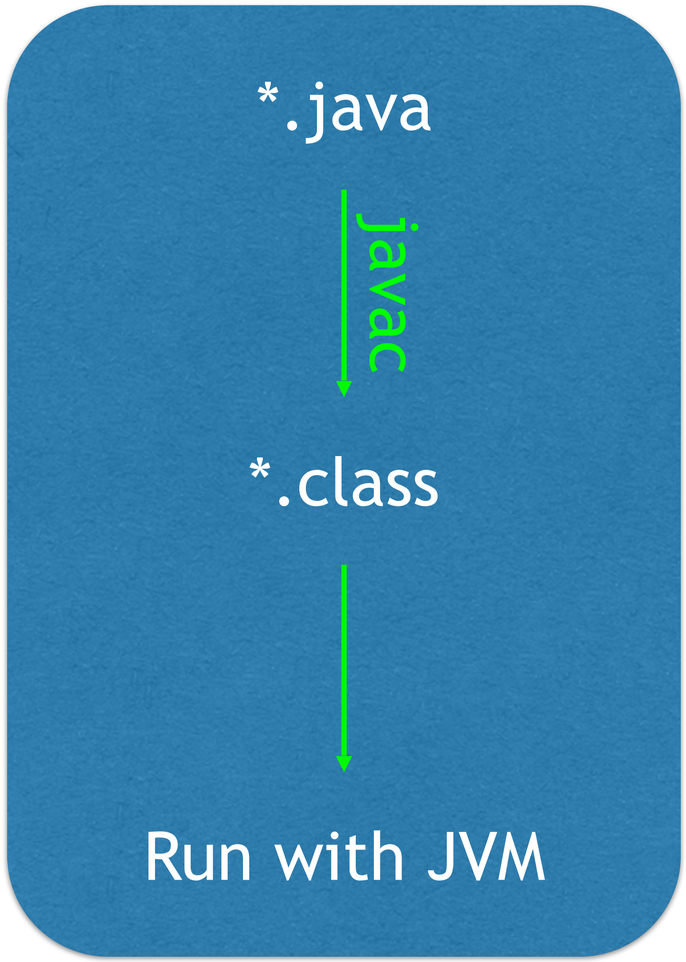


Java Model

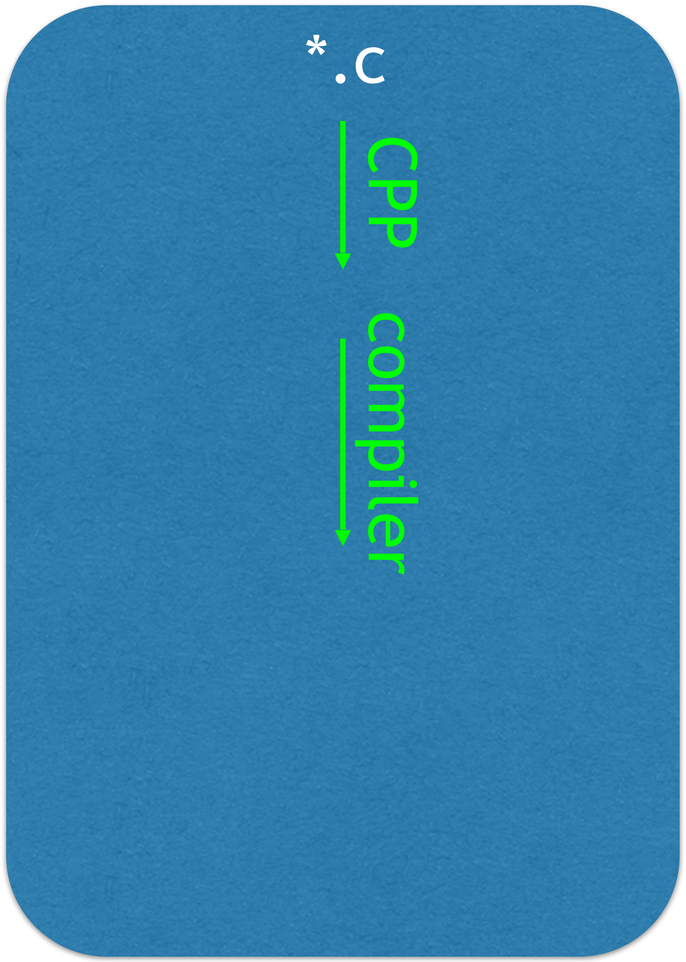


C Model

Compilation

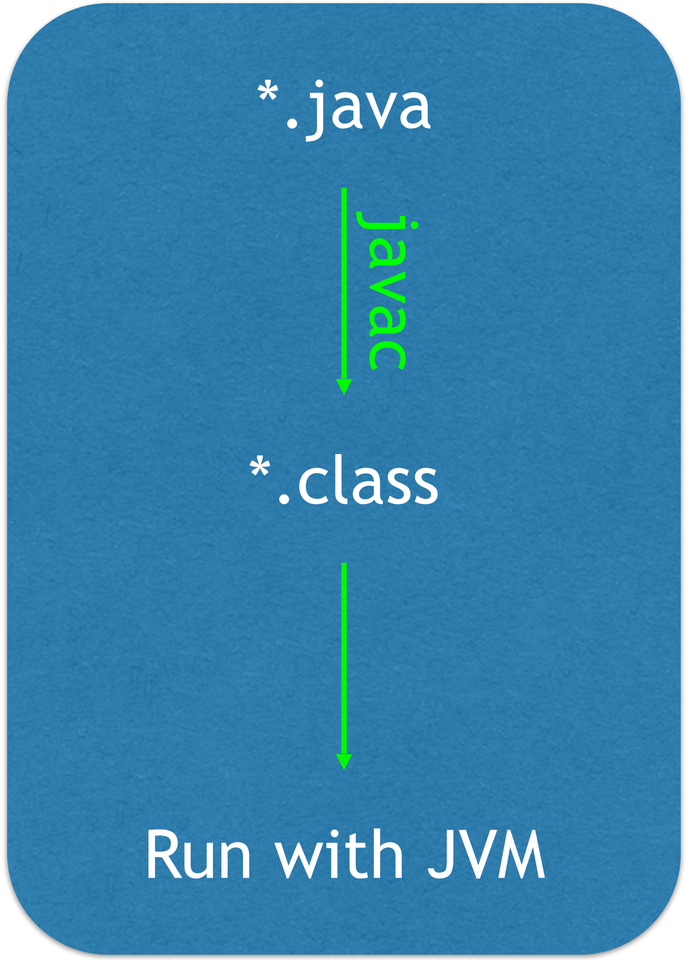


Java Model

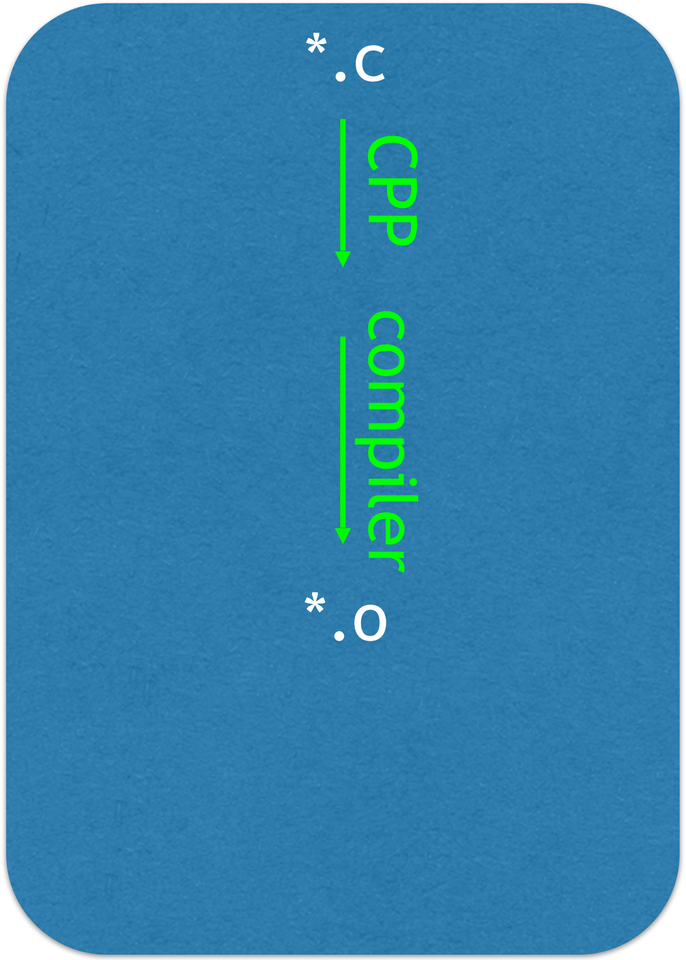


C Model

Compilation

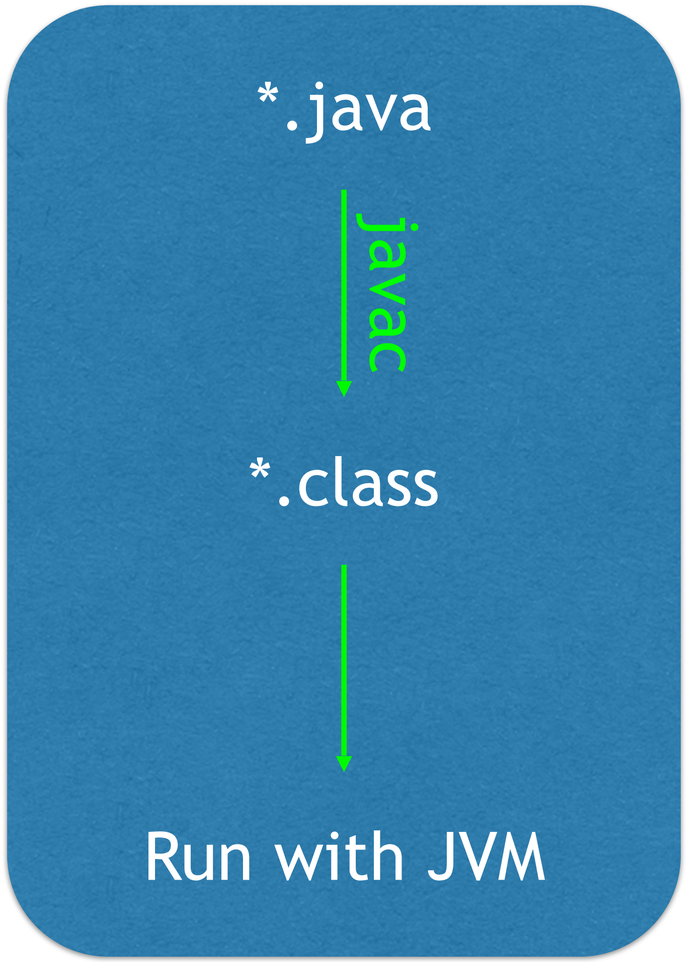


Java Model

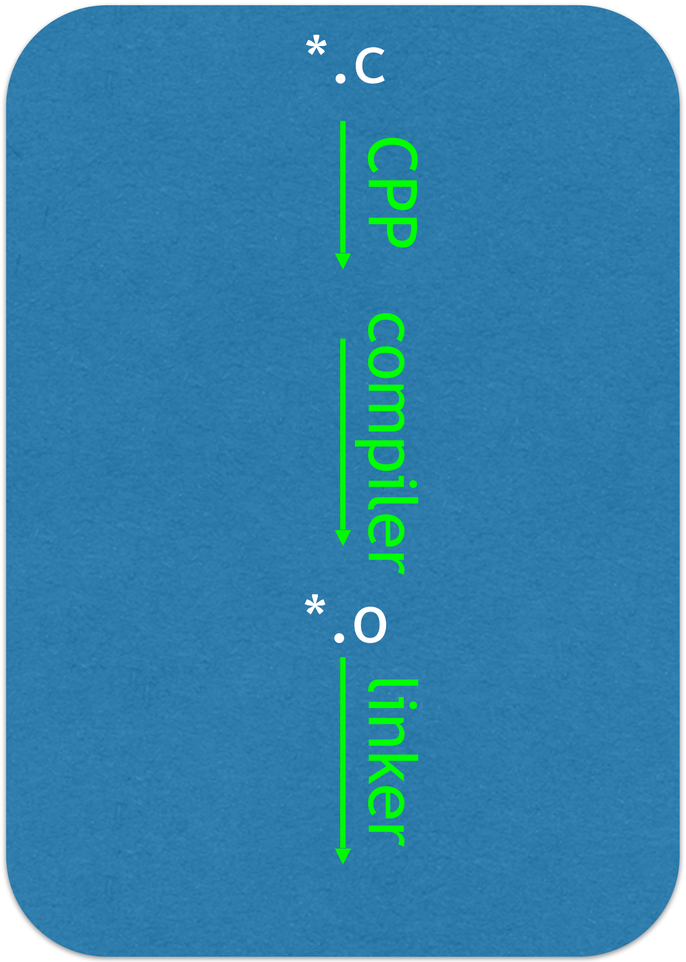


C Model

Compilation

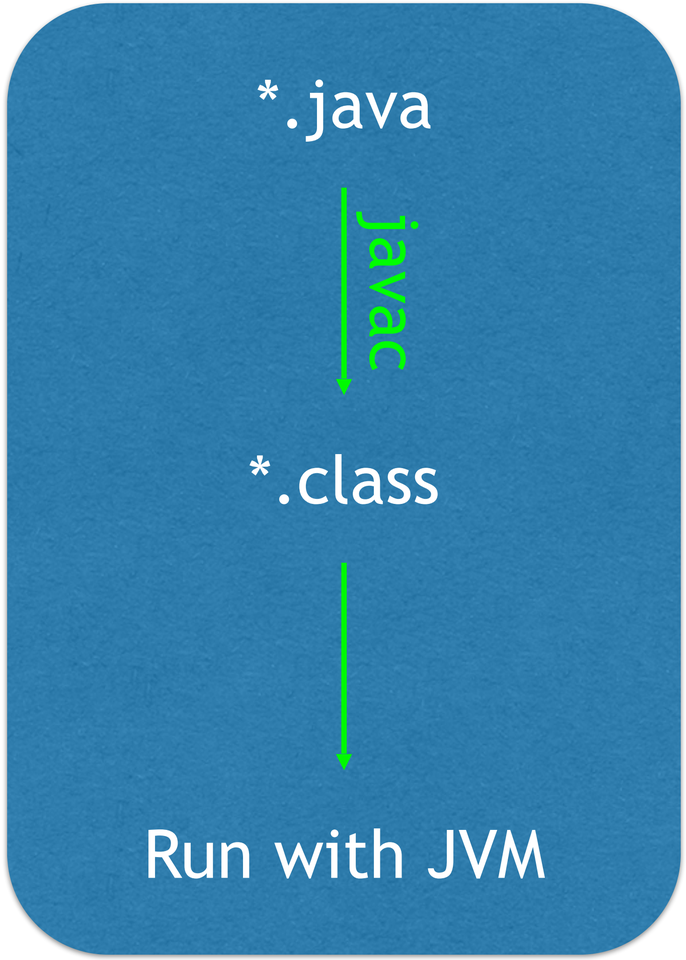


Java Model

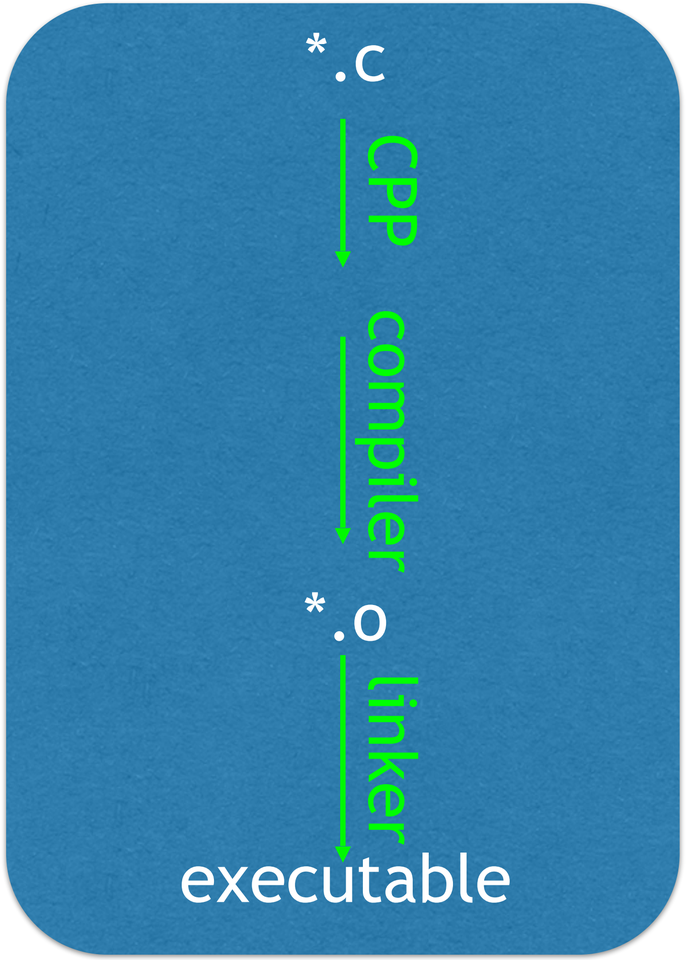


C Model

Compilation



Java Model



C Model

Types of files

Source files (.c extension)

- Include header files; lets the source file use functions declared in header
- Define functions and global variables
- Compiled to object files

Header files (.h extension)

- Declare (but typically not define) functions and global variables
- Standard library's functions are declared in system header files
- Functions used by multiple source files are declared in some header file

Object files (.o extension)

- Linked together into the executable by the linker

C Preprocessor Directives

C Preprocessor Directives

`#include` — literal inclusion of a file

- `#include <foo.h>`
- `#include "foo.h"`
- No (meaningful) differences between `<foo.h>` and `"foo.h"`

C Preprocessor Directives

`#include` — literal inclusion of a file

- `#include <foo.h>`
- `#include "foo.h"`
- No (meaningful) differences between `<foo.h>` and `"foo.h"`

`#define foo bar` — *literal replacement* of “`foo`” with “`bar`”

- Useful for symbolic constants (and other things)
- Use UPPERCASE for constants
  - Usually these are at the top of the file
  - `#define NUM_WIDGETS 20`

In C, a function must be *declared* (or defined) before the point in the source file it is called. (I.e., before calling a `int fun(double x)` function, `fun` must be declared or defined.)

How can `fun` be called from two source files, `foo.c` and `bar.c`?

A. Declare fun at the top of both `foo.c` and `bar.c`

B. Define fun at the top of both `foo.c` and `bar.c`

C. Declare fun in a header file and `#include` that file in `foo.c` and `bar.c`

D. Define fun in a header file and `#include` that file in `foo.c` and `bar.c`

E. None of the above

Consider the two files header_file.h and source_file.c shown to the right.

What is the value of x after the first line of main?

```c
// In header_file.h
#define BAR 10
#define FOO BAR+1

// In source_file.c
#include "header_file.h"
int main(void) {
  int x = FOO * 2;
  /* ... */
}
```

A. 10

B. 12

C. 20

D. 22

E. It's an error

Command line parameters

```
1  // stdio.h contains printf's declaration.
2  #include <stdio.h>
3
4  // argc is like Bash's $# (but off-by-one)
5  // argv[0] is like $0
6  // argv[1], ..., argv[argc-1] is like $1, $2 ...
7  int main(int argc, char **argv) {
8    for (int idx = 0; idx < argc; ++idx) {
9      // %d means print an integer,
10     // %s means print a string
11     printf("%d: %s\n", idx, argv[idx]);
12   }
13   return 0;
14 }
```

Command line parameters

```
$ ./arguments 'First argument' second third etc.
0: ./arguments
1: First argument
2: second
3: third
4: etc.
```

Basic types

class		systematic name	other name
integers	unsigned	`_Bool`	`bool`
		`unsigned char`	
		`unsigned short`	
		`unsigned int`	`unsigned`
		`unsigned long`	
		`unsigned long long`	
	[un]signed	`char`	
	signed	`signed char`	
		`signed short`	`short`
		`signed int`	`signed` or `int`
		`signed long`	`long`
		`signed long long`	`long long`
floating point	real	`float`	
		`double`	
		`long double`	
	complex	`float _Complex`	`float complex`
		`double _Complex`	`double complex`
		`long double _Complex`	`long double complex`

Integer type sizes

Integer type sizes

`sizeof(type)` is the number of bytes a variable of `type` has

Integer type sizes

`sizeof(type)` is the number of bytes a variable of `type` has

```
1 = sizeof(char) ≤ sizeof(short) ≤ sizeof(int)
  ≤ sizeof(long) ≤ sizeof(long long)
```

Integer type sizes

`sizeof(type)` is the number of bytes a variable of `type` has

```
1 = sizeof(char) ≤ sizeof(short) ≤ sizeof(int)
  ≤ sizeof(long) ≤ sizeof(long long)

sizeof(type) = sizeof(signed type) = sizeof(unsigned type)
```

Integer type sizes

`sizeof(type)` is the number of bytes a variable of `type` has

```
1 = sizeof(char) ≤ sizeof(short) ≤ sizeof(int)
  ≤ sizeof(long) ≤ sizeof(long long)
```

```
sizeof(type) = sizeof(signed type) = sizeof(unsigned type)
```

`sizeof(bool)` is implementation defined

Integer type sizes

`sizeof(type)` is the number of bytes a variable of `type` has

```
1 = sizeof(char) ≤ sizeof(short) ≤ sizeof(int)
  ≤ sizeof(long) ≤ sizeof(long long)
```

```
sizeof(type) = sizeof(signed type) = sizeof(unsigned type)
```

`sizeof(bool)` is implementation defined

A byte isn't always 8 bits! (But it is on most systems.)

In-class exercise

https://checkoway.net/teaching/cs241/2020-spring/exercises/Lecture-08.html

Grab a laptop and a partner and try to get as much of that done as you can!