CS142B Language Processor Construction

Overview

Yeoul Na

UCI

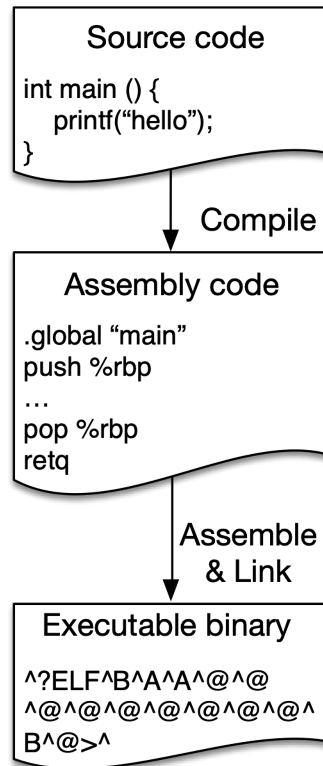
April 2, 2019

General Information

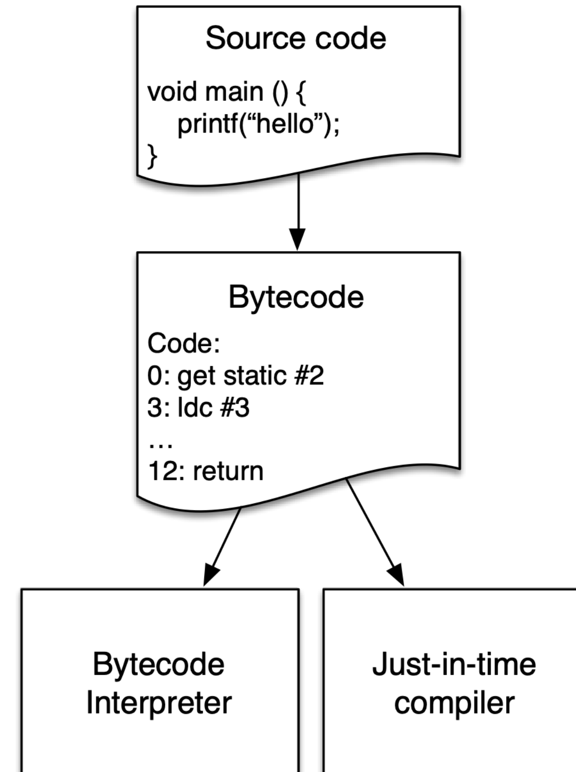
- A heavy project class
- Drop the class immediately
  - If you don’t like programming
  - If you take more than one project classes
- Prerequisites
  - Background knowledge about compilers
  - C/C++ programming skills
- Expectations
  - Research yourself
  - Don’t rely on lectures!

Reminder on Language Processing

Compilation



Interpretation



Haven't we solved the problem yet?

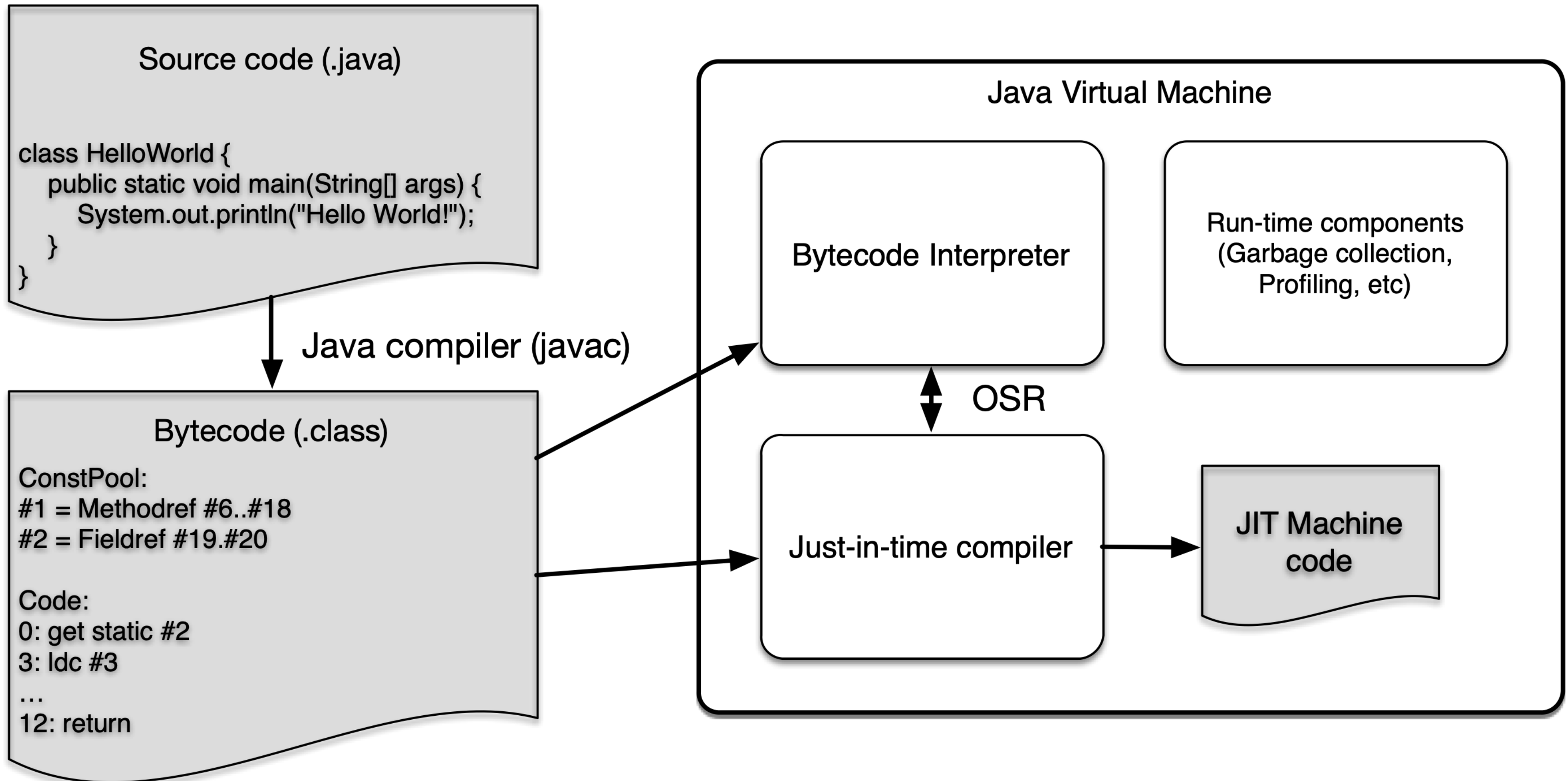
- New languages: Rust, Go, …
- New hardware: GPU, FPGA, Neural Processor, …
- Cybersecurity
- Technology innovation
    - Big Data
    - Internet Of Things (IoT)
    - Artificial Intelligence (AI)

- Lack of compiler experts

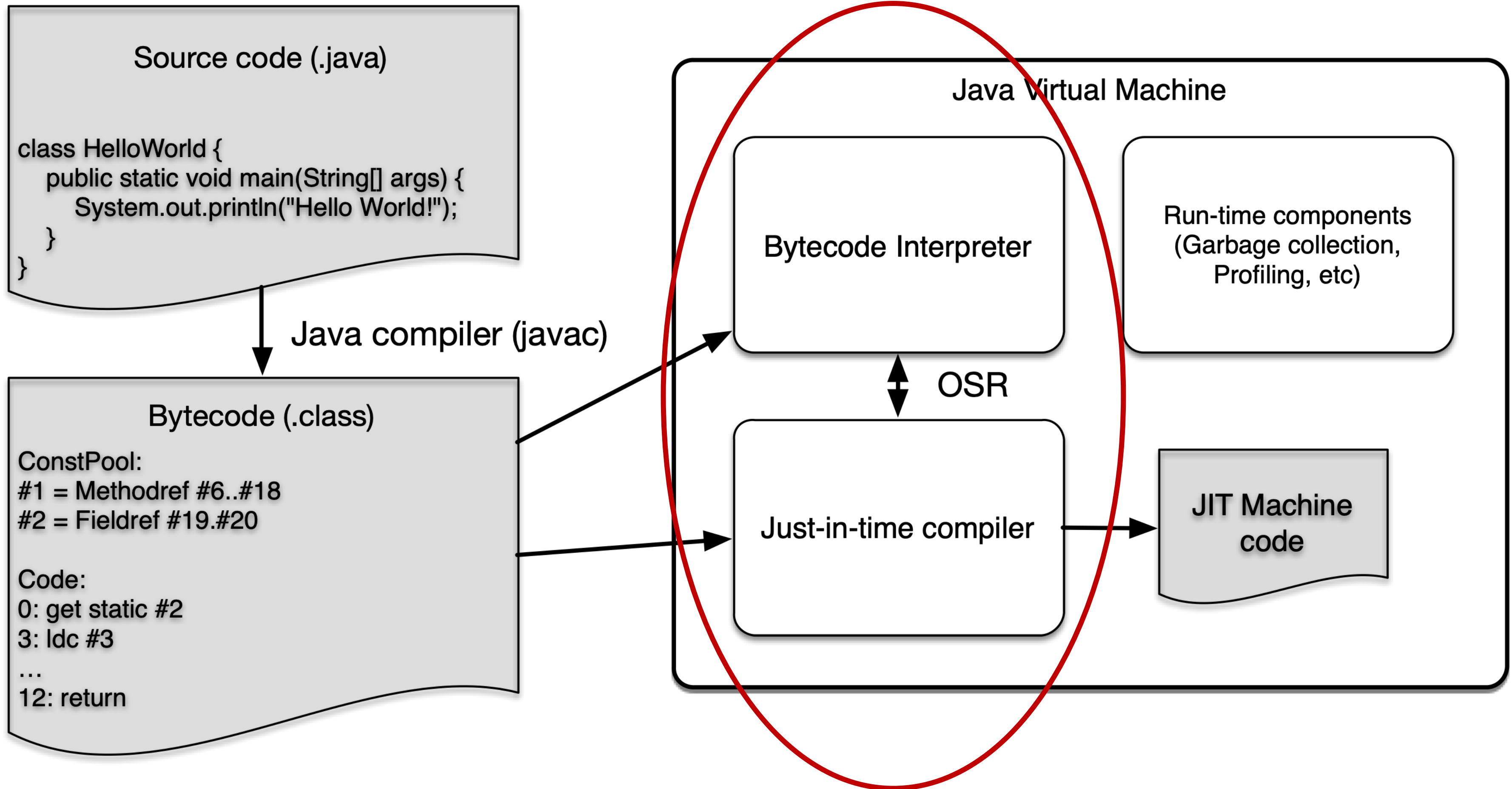
About this course

- Language processor construction
  - Implement a mini Java Virtual Machine (JVM)

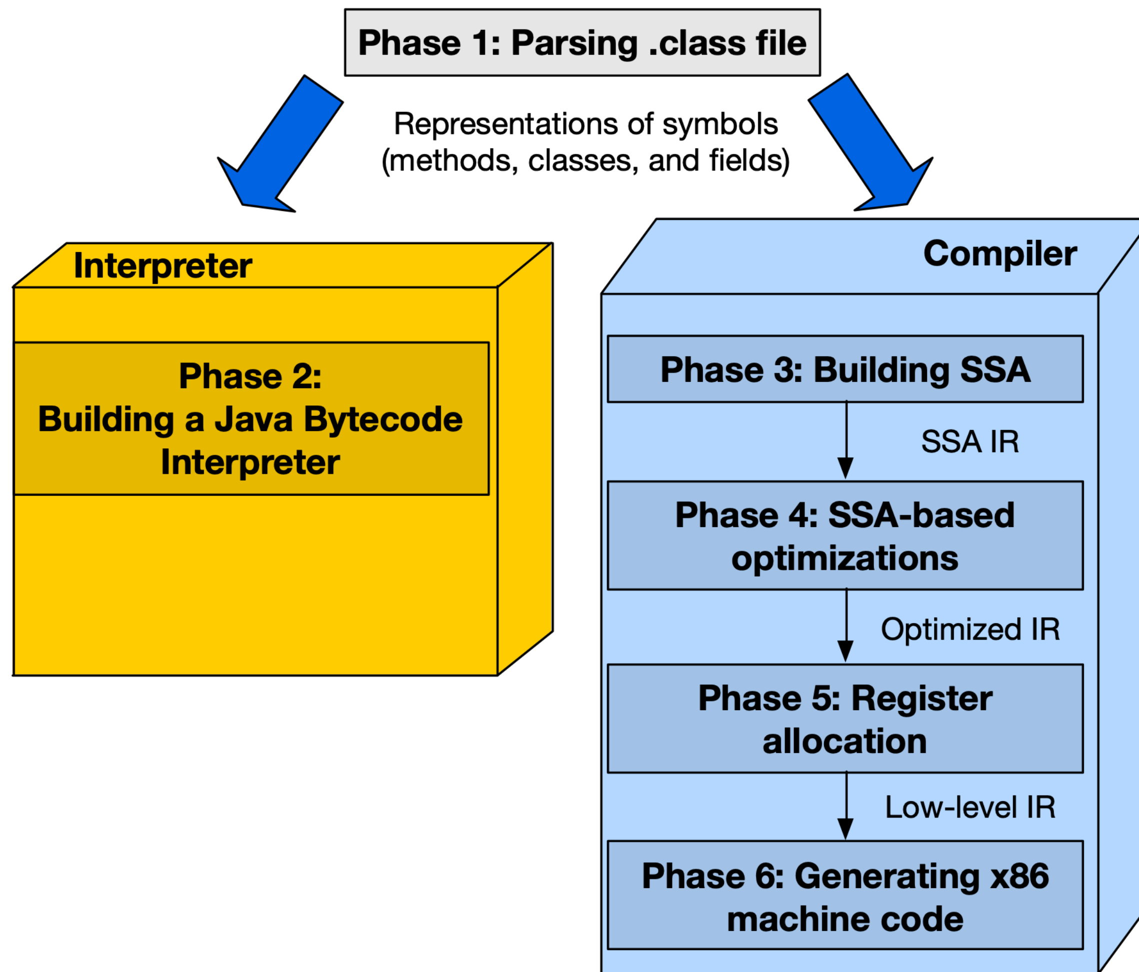
Java Language Processing



Java Language Processing



Project Phases



Expected Outcome

- After you this course, you will have hands-on experience with
  - What is inside a JVM
  - How interpreter works
  - Dataflow analysis and optimizations
  - X86 assembler
  - How to implement a real-world compiler
- You will become very proud of what you just implemented

Lecture Schedule

Week	Date	Lectures	
		Tuesday	Thursday
1	Apr 2 / Apr 4	**Overview & Parsing .class file**	*No lecture – working on the project*
2	Apr 9 / Apr 11	**Bytecode interpreter**	*No lecture – working on the project*
3	Apr 16 / Apr 18	**Single-static assignment (SSA)**	*No lecture – working on the project*
4	Apr 23 / Apr 25	*No lecture – working on the project*	*No lecture – working on the project*
5	Apr 30 / May 2	**SSA-based optimizations**	*No lecture – working on the project*
6	May 7 / May 9	*No lecture – working on the project*	*No lecture – working on the project*
7	May 14 / May 16	**Register allocation**	*No lecture – working on the project*
8	May 21 / May 23	*No lecture – working on the project*	*No lecture – working on the project*
9	May 28 / May 30	**x86 machine code generation**	*No lecture – working on the project*
10	Jun 4 / Jun 6	**Advanced Topic - Security**	No lecture – working on the project
Final	Jun 8 ~ Jun 13	**Project Demo (appointment-base)**	

Grading Policy

- You will receive at least a *B* if your compiler can parse and interpret the bytecode and generate SSA.
- You will receive at least an *A* if your compiler can generate x86 machine code and passes all my test cases.
- You will receive an *A+* if your compiler does everything and has one additional dataflow optimization implemented.

Resources

- Office hour
  - CS Building 444
  - Thursdays 9:30-10:50, or Make an appointment
- Our slack channel - UCI-CS142b.slack.com
  - Announcements
  - Help each other (Q&A)
- Resource updates
  - https://www.ics.uci.edu/~yeouln/course/cs142b

Overview of the *.class* file

- Contain Java bytecode that can be executed on the JVM
- Platform independent
- In binary format (*not* in ASCII)
    - be careful when reading a binary format
- Include 10 basic sections

Sections in the *.class* file

Sections	Description
Magic number	0xCAFEBABE
Class file version info	The minor and major versions
Constant pool	Pool of constants for the class
Access flags	abstract, static, etc.
This class	the name of the current class
Super class	the name of the super class
Interfaces	any interfaces in the class
Fields	any fields in the class
Methods	any methods in the class
Attributes	any attributes of the class (e.g., the source file name)

.class file – high-level representation

https://en.wikipedia.org/wiki/Java_class_file

```
struct Class_File_Format {
   u4 magic_number;

   u2 minor_version;
   u2 major_version;

   u2 constant_pool_count;

   cp_info constant_pool[constant_pool_count - 1];

   u2 access_flags;

   u2 this_class;
   u2 super_class;

   u2 interfaces_count;

   u2 interfaces[interfaces_count];

   u2 fields_count;
   field_info fields[fields_count];

   u2 methods_count;
   method_info methods[methods_count];

   u2 attributes_count;
   attribute_info attributes[attributes_count];
}
```

Example

- vim Test.class -> :%!xxd

```
00000000: cafe babe 0000 0034 001e 0a00 0600 1109
00000010: 0012 0013 0a00 1400 150a 0005 0016 0700
```

  - 0: Magic number (u4) - 0xcafebabe
  - 4: Minor version (u2) - 0x0000 (0)
  - 6: Major version (u2) – 0x0034 (52)
  - 8: Constant pool count (u2) – 0x001e (30)
  - …

Constant pool

- Constant pool entry format:

```
cp_info {
    u1 tag;
    u1 info[];
}
```

  - Tag indicates the type of constant pool entry
  - Size of info[] varies across the type of entry

Tag = 10 (MethodRef)

```
CONSTANT_Methodref_info {
    u1 tag;
    u2 class_index;
    u2 name_and_type_index;
}
```

Tag = 3 (Integer)

```
CONSTANT_Integer_info {
    u1 tag;
    u4 bytes; // big-endian
}
```

https://docs.oracle.com/javase/specs/jvms/se12/html/jvms-4.html#jvms-4.4

Example: your representation of a constant pool entry

```
class cp_info {
  uint8_t tag;
  uint8_t info[];
public:
  uint8_t getInfo(int i) {
    return info[i];
  }
};

class cp_methodref_info : cp_info {
public:
   uint16_t get_class_index() {
        return (getInfo(0) << 8)|getInfo(1);
   }
};
```

Your internal representation

```
class ClassInfo {
    MethodInfo [] methods;
    Field[] fields;
    Attribute[] attributes;
    ClassInfo[] superclasses;
    …
};

Class MethodInfo {
   Qualifier[] qualifiers;
   uint8_t [] bytecodes;
};
```

Where is bytecode of a method?

- Read attribute_info in the method
- Consult the constant pool[attribute_name_index] to get the name string, and if the name is “Code”.

```
method_info {
    u2             access_flags;
    u2             name_index;
    u2             descriptor_index;
    u2             attributes_count;
    attribute_info attributes[attributes_count];}

attribute_info {
    u2 attribute_name_index;
    u4 attribute_length;
    u1 info[attribute_length];
}
```

Code attribute

- Ignore the exception table and the attribute

```
Code_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 max_stack;
    u2 max_locals;
    u4 code_length;
    u1 code[code_length];
    u2 exception_table_length;
    {   u2 start_pc;
        u2 end_pc;
        u2 handler_pc;
        u2 catch_type;
    } exception_table[exception_table_length];
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

Make it simple!

- Bytecode has too much information
- We are only interested in ConstantPool, ThisClass, Methods (and Method Parameters, and Code)
- You can just skip and ignore all other information
  - Ignore exceptions and run-time components
  - Ignore super classes
  - Start with what you need!

Again, research your self.
Don’t solely rely on the lecture.