Week 5 Discussion

CS 131 Section 1B
29 April 2022
Danning Yu

Announcements

- HW3 released, due 5/4
  - Starter code posted under Week 5 on BruinLearn
- Homeworks should be submitted on BruinLearn, under Assignments
- Before submitting
  - Make sure your code compiles on SEASnet server
  - Make sure your function signatures are correct
  - Follow all instructions and specifications
  - Do not submit files in a .zip unless told to do so
- Help and starter code from past TAs
  - https://github.com/CS131-TA-team

Basic Java

Pictures and diagrams borrowed from Boyan Ding and others

Object Oriented Programming

- Main idea: objects with methods and fields
    - Methods and fields are functions and variables that belong to the object, and encapsulated within
    - Object of the same class share same fields and methods
- Most popular programming paradigm
    - Java, C++, C#, Python, PHP, JavaScript, Ruby, Objective-C, Swift, Scala, Common Lisp, Smalltalk, …
    - i.e. Most of the popular languages today
- Possible benefits
    - Modularity
    - Information-hiding
    - Code reuse
    - Pluggability and ease of debugging
    - And more

Classes and Interfaces

- Class: template for an object
  - Object is an instance of a class
  - e.g. We can have multiple Bicycle objects that function the same way, but can be moving at different speed etc.
- All objects created using the same class will have the same methods/fields
- Interface: a description of what needs to be implemented
  - Multiple classes can implement the same interface
  - In Java, a list of functions
  - Allows for separation of API from implementation

Alan Kay's Definition of OOP

- Everything is an object
    - Numbers, classes, functions, ...
- Objects communicate by sending/receiving messages
    - Think of biological cells communicating
- Objects have their own memory
- Every object is an instance of some class
- All objects of the same type can receive the same messages

Some of these do not apply to most modern OOP languages!

Java

- General-purpose, object-oriented language
- One of the most popular programming languages
- Code compiled into bytecode and runs on a virtual machine
  - What are the pros and cons of this?
- Popular IDEs include Eclipse, Intellij IDEA
  - We don’t require usage of IDE, you can use any text editor for your homework

Java: Hello World

- `HelloWorld.java`

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

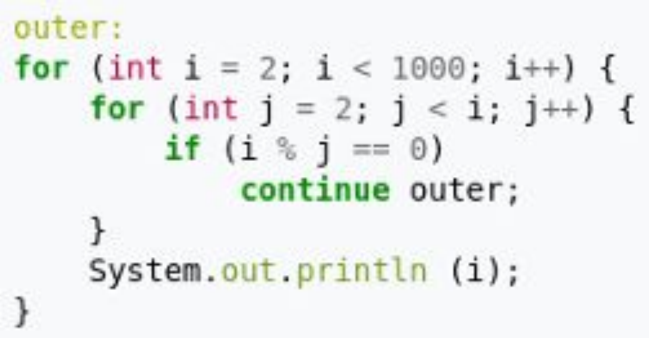
- How to compile
  - `javac HelloWorld.java`
  - Generates `HelloWorld.class` containing bytecode
- Running
  - `java HelloWorld`
  - Note: use class name, not file name

Files in Java

- `MyClass.java`: code for `MyClass`
- `MyClass.class`: bytecode for `MyClass` (compiled from `Myclass.java`)
- `Foo.jar`: Java Archive File
  - Is really just a ZIP archive
  - Often used to package whole compiled application with resources, configuration, etc
  - Will use this to package source files for Homework 3

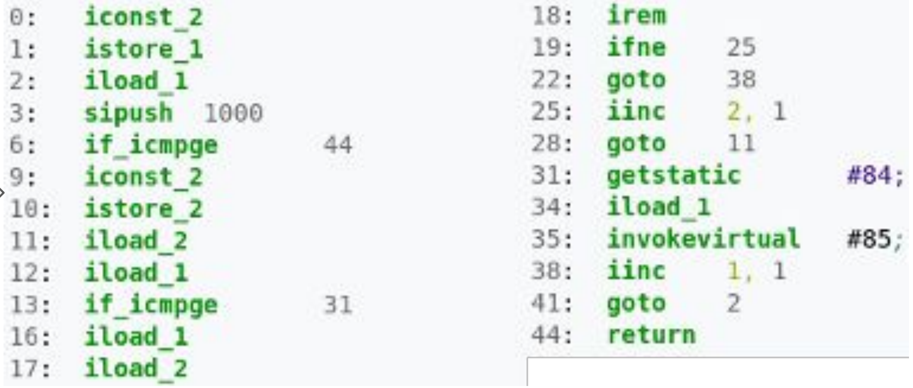
Java Bytecode

- A intermediate form between compiled and interpreted code
  - Platform independence of interpreted code
  - Better performance than interpreted code



```
outer:
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

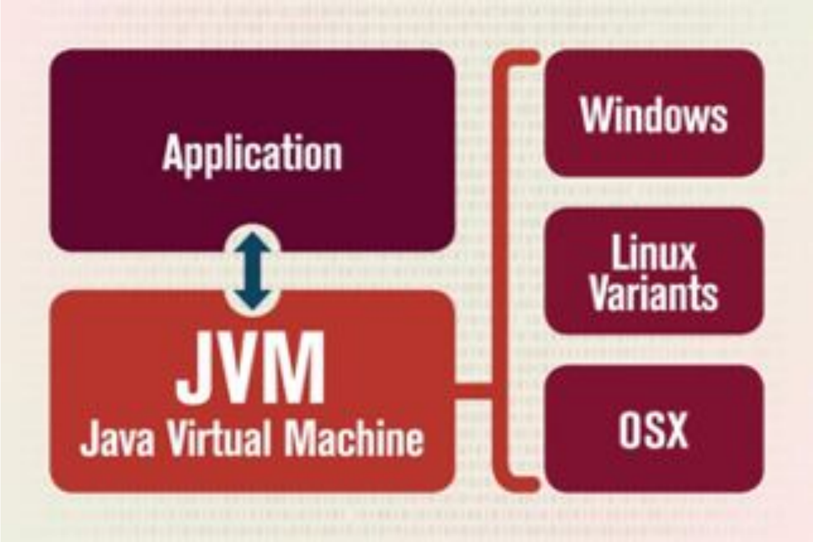
javac



```
0:    iconst_2
1:    istore_1
2:    iload_1
3:    sipush  1000
6:    if_icmpge       44
9:    iconst_2
10:   istore_2
11:   iload_2
12:   iload_1
13:   if_icmpge       31
16:   iload_1
17:   iload_2
18:   irem
19:   ifne    25
22:   goto    38
25:   iinc    2, 1
28:   goto    11
31:   getstatic       #84;
34:   iload_1
35:   invokevirtual   #85;
38:   iinc    1, 1
41:   goto    2
44:   return
```

Java Virtual Machine (JVM)

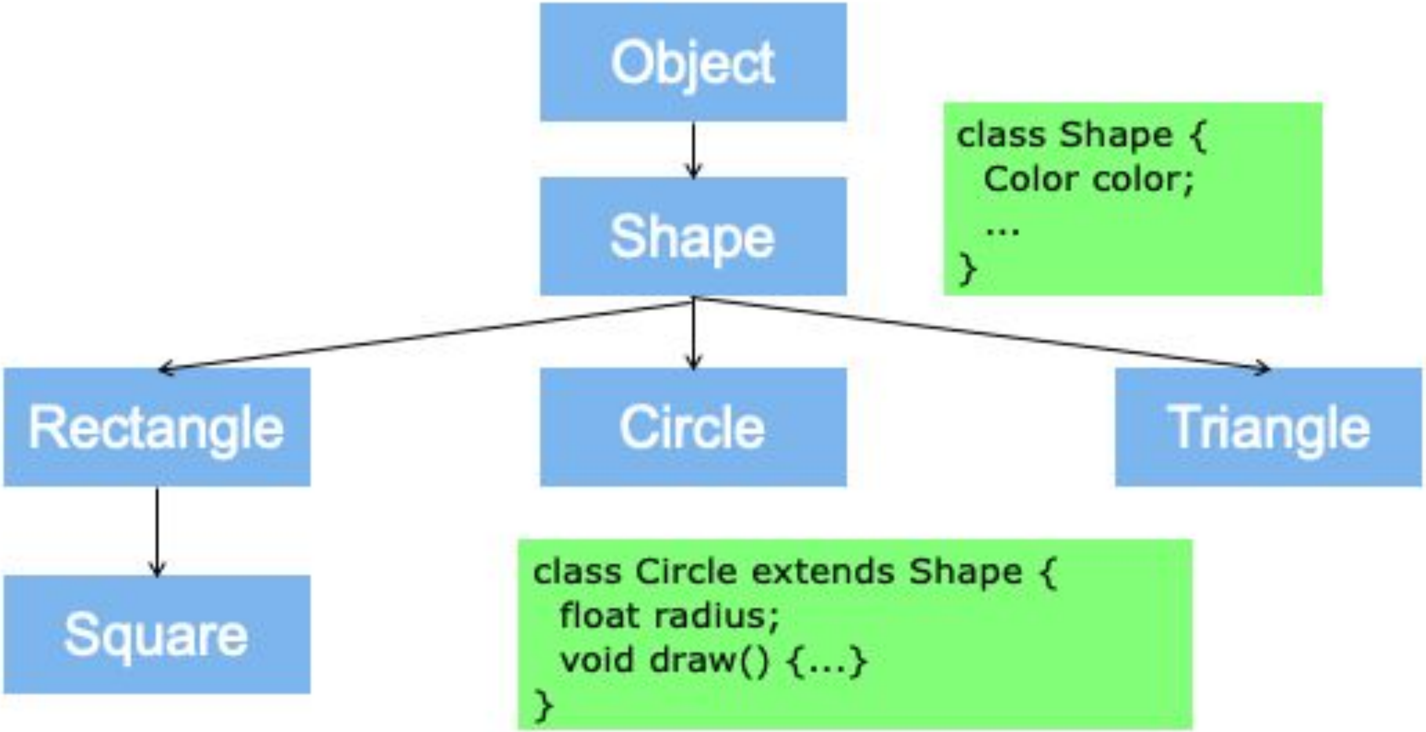
- Runs bytecode generated by a Java compiler
- Provides separation of code and operating system
  - Allows Java code to run on a variety of OSes
- JVM provides garbage collection, just-in-time compilation (JIT), etc
- Multiple implementations
  - Reference implementation (OpenJDK) provided by Oracle



OOP In Java

- Abstraction
- Encapsulation
  - Binding data with code that manipulates it
  - Access modifiers: `public/protected/private`
- Inheritance
  - An object may acquire some/all property of another object
- Polymorphism
  - One method can have multiple implementations, usage decided at runtime

Inheritance in Java



```
class Shape {
  Color color;
  ...
}
```

```
class Circle extends Shape {
  float radius;
  void draw() {...}
}
```

Inheritance in Java

```
class Shape {
    void draw() { /* do nothing */ }
}
class Rectangle extends Shape {
    void draw() { /* draw a rectangle */
}
}
class Circle extends Shape {
    void draw() { /* draw a circle */ }
}
class Triangle extends Shape {
    void draw() { /* draw a triangle */ }
}
```

```
Triangle a = new Triangle();
/* draws a triangle */
a.draw();

Shape b = a;
/* draws a triangle */
b.draw();

b = new Circle();
/* draws a circle */
b.draw();
```

Inheritance in Java

- Which of the following are allowed?

```
Square a = new Square();
Shape b = a;
```

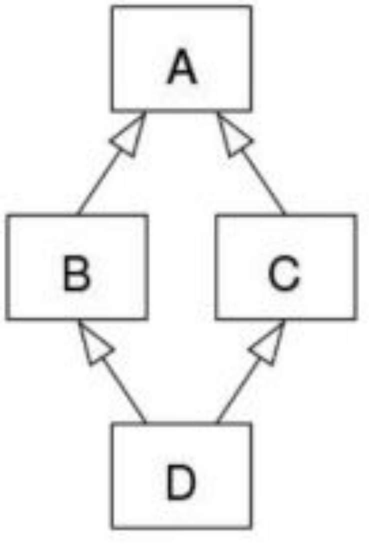
```
Shape a = new Shape();
Square b = a;
```

```
Shape a = new Square();
Square b = a;
```

- Left: allowed
- Middle: not allowed
  - `Shape` does not have the same methods or fields as `Square`
- Right: not allowed
  - Need to cast it

Inheritance in Java

- Multiple inheritance is not allowed in Java
  - Why?



Diamond Problem

Interface

- Defines what a class must be able to do, not how to do it
    - Can’t be instantiated, should only be implemented by classes
- One class can implement multiple interfaces

```
interface Vehicle {
  public void increaseSpeed();
  public void decreaseSpeed();
  public void turnLeft();
  public void turnRight();
}
```

```
class Car implements Vehicle {
  public void increaseSpeed() {
    /* Press accelerator */
  }
  public void decreaseSpeed() {
    /* Press brake pedal */
  }
  /* other implementations */
}
```

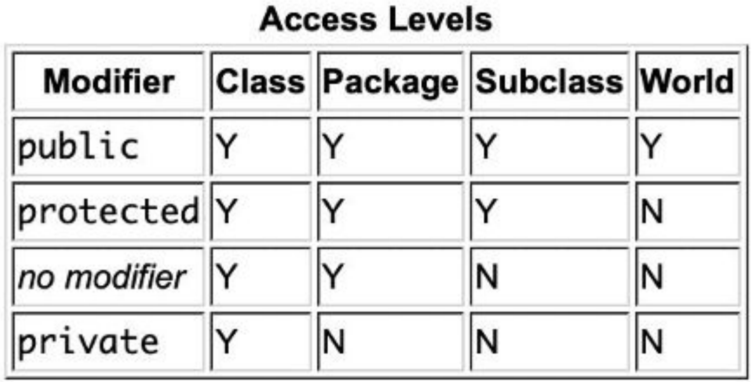
Abstract Classes

- Combination of a class and an interface
  - Similar to abstract class in C++ (pure virtual function)
- Objects of an abstract class cannot be created, can only be inherited
- `abstract` method: no implementation, must be implemented by subclasses

```
abstract class Shape {
    abstract void draw();
    void setColor(Color c) {
        /* set color */
    }
}
```

Access Modifiers

- Controls who can access an object’s methods/fields
  - In general, start with `private` and make fields more visible only when necessary
- Classes also have access modifiers: `public` or no modifier (makes it package private)
- Package-private: can access within package but not outside
  - Related Java files are typically grouped together into a package

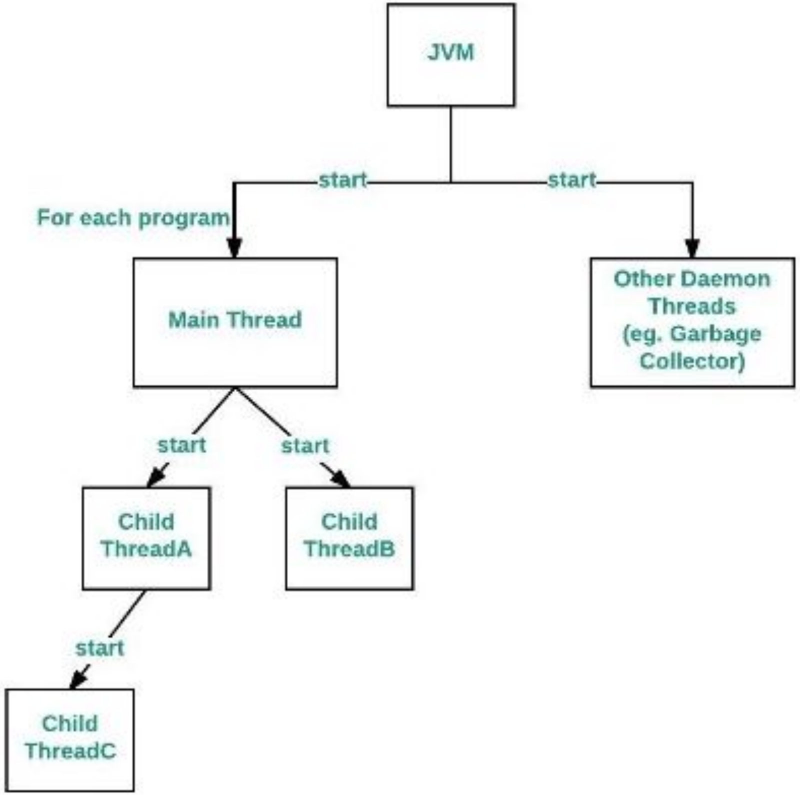


Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Java Memory Model

Threads in Java



Creating and Using Threads

```
public class MyRunnable implements Runnable {
  public void run() {
    System.out.println("MyRunnable - START ");
    // Do some heavy processing here
    System.out.println("MyRunnable - END ");
  }
}

// In your main method:
Thread t1 = new Thread(new MyRunnable());
Thread t2 = new Thread(new MyRunnable());
t1.start(); // Start executing thread 1
t2.start(); // Start executing thread 2
t1.join(); // Wait for thread 1 to finish
t2.join(); // Wait for thread 2 to finish
```

Java Memory Model

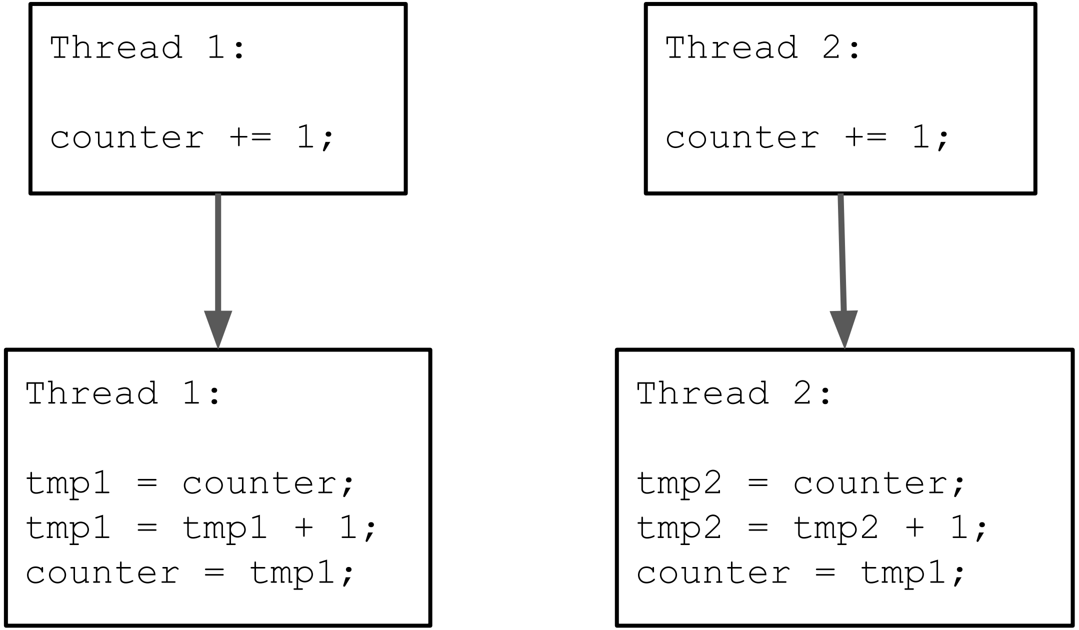
- Defines how threads interact with memory
- Defines what code reorderings are legal for the compiler and processor to carry out
- “As-if-serial” semantics used within one thread
  - Compiler can change your code in any way as long as the result of execution is the same
- e.g. in the following code, `x` and `y` are both initially 0
  - Java may freely reorder the assignment on the left, but not on the right

```
x = 2;
y = 1;
System.out.format(“%d %d\n”, x, y);
```

```
x = 2;
System.out.format(“%d %d\n”, x, y);
y = 1;
System.out.format(“%d %d\n”, x, y);
```

- With multiple threads are runnning, the situation gets complicated

Race Conditions



```
Thread 1:

counter += 1;
```

```
Thread 2:

counter += 1;
```

```
Thread 1:

tmp1 = counter;
tmp1 = tmp1 + 1;
counter = tmp1;
```

```
Thread 2:

tmp2 = counter;
tmp2 = tmp2 + 1;
counter = tmp1;
```

`synchronized` Keyword

- Each object has a lock
- This keyword enforces exclusive access
  - Only one thread can enter a `synchronized` method in one object at once
- Happens-before relationship
  - Everything that one thread did while in a synchroized block will be visible to the next thread entering a synchroized block
- A thread can call any other synchroized methods while it holds the lock

synchronized Keyword: Example

```
public class SynchronizedCounter {
    private int c = 0;

    public synchronized void increment() {
        c++;
    }

    public synchronized void decrement() {
        c--;
    }

    public synchronized int value() {
        return c;
    }
}
```

`synchronized` Keyword: Example

- Synchronized can also be used with smaller blocks of code
- Avoid blocking others when it’s not necessary

```
public class SynchronizedCounter {
    private int c = 0;

    public void incrementAndWork() {
        // ... computation here ....
        synchronized(this) {
            c++;
        }
        // ... computation here ....
    }
}
```

`synchronized` Keyword

- Any object can be used as the lock for `synchronized`

```
public class MyClass {
  private int c1 = 0;
  private int c2 = 0;
  private Object lock1 = new Object();
  private Object lock2 = new Object();
  public void inc1() {
    synchronized(lock1) {
    c1++;
    }
  }
  public void inc2() {
    synchronized(lock2) {
    c2++;
    }
  }
}
```

`volatile` Keyword

- Guarantees that other threads will see the changes immediately
  - `volatile` access can not be reordered relative to other reads/writes
  - Effectively serves as a memory barrier
- Excellent explanation with details [found at this link]()

```
Thread 1:

x = 5;
done = true;
```

```
Thread 2:

while (!done) {}
System.out.println(x);
```

- Without `volatile`, the printed value may not always be 5
- If `done` is defined as `volatile`, then `x` will always be printed as 5

Atomic Operations

- Another option for preventing race conditions
- Atomic operations: code is translated into assembly instructions that guarantee the update value is visible to all threads without race conditions
  - Typically means a single assembly instruction, but sometimes more
  - Other threads do not see intermidiate states, only final state
- Atomic package `java.util.concurrent.atomic` provides data types with atomic operations
- `AtomicInteger` can be used to perform `cnt++` as an atomic operation

```
AtomicInteger cnt = new AtomicInteger(5);
cnt.incrementAndGet();
```

Optional: C++ Memory Model

- C++ did not have a memory model until C++11
  - Before then, multithreading behavior was technically unspecified!
- Great talk on the C++ memory model
  - Part 1: https://www.youtube.com/watch?v=A8eCGOqgvH4
  - Part 2: https://www.youtube.com/watch?v=KeLBd2EJLOU

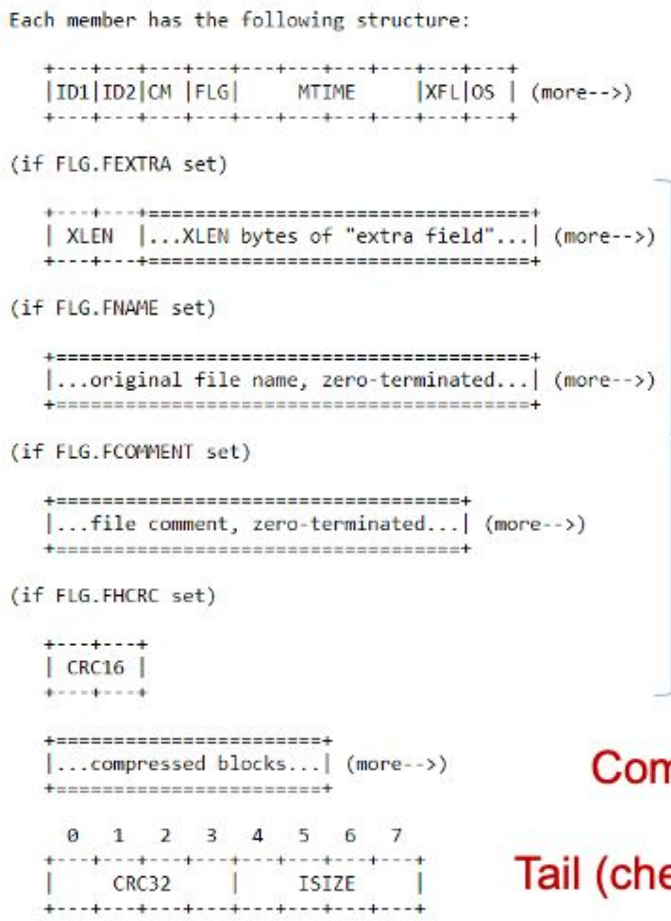
Homework 3

Introduction

- Multithreaded gzip compression
  - Implementing pigz in Java
  - No need to implement the actual compression algorithm, but rather, make the existing compression algorithm multithreaded
- Can leverage the ideas of an existing implementation called MessAdmin
- Also use starter code on BruinLearn to help you
- Test and compare 4 programs: gzip and pigz in Linux; Pigzj on JVM and native version (compiled with native-image)

GZip Compression

- gzip: a stream compression format based on DEFLATE
- Input and output are binary streams
- Commonly used for file compression
  - `.gz` extension
  - Note: `.tar` is just a tarball: a collection of files (with no compression) usage: e.g. file compression
- Format specified in RFC 1952
- A gzip file can have one or more members shown on the right
- Java implements this algorithm and other compression algorithms in `java.util.zip`



```
Each member has the following structure:

+---+---+---+---+---+---+---+---+---+---+
|ID1|ID2|CM |FLG|     MTIME     |XFL|OS | (more-->)
+---+---+---+---+---+---+---+---+---+---+

(if FLG.FEXTRA set)

+---+---+=================================+
| XLEN  |...XLEN bytes of "extra field"...| (more-->)
+---+---+=================================+

(if FLG.FNAME set)

+=========================================+
|...original file name, zero-terminated...| (more-->)
+=========================================+

(if FLG.FCOMMENT set)

+===================================+
|...file comment, zero-terminated...| (more-->)
+===================================+

(if FLG.FHCRC set)

+---+---+
| CRC16 |
+---+---+

+=======================+
|...compressed blocks...| (more-->)
+=======================+

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|     CRC32     |     ISIZE     |
+---+---+---+---+---+---+---+---+
```

Header

Header extensions

Compressed Data

Tail (checksum + size)

GZip Compression in Java

- Implementing a simple gzip equivalent in Java
- Take input from stdin, writes output to stdout
  - Equivalent to `gzip -c -`

```java
import java.io.IOException;
import java.util.zip.GZIPOutputStream;

public class SimpleGZip {
  public static void main(String[] args) throws IOException {
    GZIPOutputStream gzout = new GZIPOutputStream(System.out);
    System.in.transferTo(gzout);
    gzout.close();
  }
}
```

Parallel GZip Compression

- A basic implementation of gzip process the file linearly
- How to parallelize compression (with P threads)
    - Break the files into P partitions, with each thread processing one partition. Then, concatenated the compressed partitions together
    - Pigz’s approach: divide input into fixed size blocks (128 KiB), and have P threads busily processing a block
        - 1 KiB = 1024 bytes
- What’s the difference? Why do we prefer the latter for the HW?
    - Allows us to handle streaming cases
    - As each thread finishes, append to the end of the file (fast)

Pigz Details

- From pigz’s manual page:
  - Checksum: “The individual check value for each chunk is also calculated in parallel... A combined check value is calculated from the individual check values.” (note: not implemented in MessAdmin)
  - Dictionary: “The input blocks, while compressed independently, have the last 32K of the previous block loaded as a preset dictionary to preserve the compression effectiveness...”

Java Environment

- Make sure you're using the correct version of Java as specified in the HW specifications
- Can also try installing GraalVM on your own machine
  - https://www.graalvm.org
  - You need the GraalVM and Native Image tool
- The native-image tool from GraalVM allows people to compile Java programs to native binaries
  - To use it, first compile Java code to class files as usual
  - Then use “native-image ClassName” to compile the java program, and you will get an executable named “classname”
  - It can be quite slow on SEASnet, use it when you are sure your java program is functioning correctly with JVM

MessAdmin

- Github link: https://github.com/MessAdmin/MessAdmin-Core
  - Relevant code can be found in `clime.messadmin.utils.compress.{gzip,impl}` packages
- The final implementation will need a class similar to `PGZIPOutputStream`
- You can learn the ideas and techniques used there, especially the following
  - The `Compressor` class, which uses `ThreadPoolExecutor` for multithreaded compression execution
  - The style of passing tasks among threads
- You're welcome to write code in your own style if you want

Homework Requirements

- The main program should be named `Pigzj`
  - This means your main class is called `Pigzj`
  - Optionally takes argument `-p processes` to specifiy the threads used, default to number of processors on the system
  - Input taken from stdin, output goes to stdout, no need for file operation
- Other requirements
  - Correctness: your compressed file should be understood by gzip/pigz
  - For full credit, output should only contain a single member
  - Ideally, the output should be byte-for-byte identical with pigz output
  - Give proper error messages for certain cases
- Full requirements found on homework page

Homework Submission

- Submit a single jar file containing
  - All the `.java` source files
  - Do not add compiled `.class` files
- A report with performance measurements and analysis
  - Plain text file under 60 kB
  - Compare the runtime of gzip, pigz, and Pigzj (both JVM and GraalVM native) with different settings
  - Try to use `strace` and see if the result can explain your findings
  - Check HW specification for full list of requirements
- Make sure to use the provided commands to sanity check your submission

Thank You