**Homework 2**

**Programming Languages Principles and Implementation**

**Instructions:**

* Due date: 10/8 (No late homework will be accepted. The solution of the homework will be posted on 10/9 after class. The midterm is shortly after.)
* This homework assignment is to be done alone or in a group of 2 students.
* Problems must be done in order.
* You need to fill out this document with your answers. Homeworks with answers only will not be accepted.
* All Java code must be written and tested in the Eclipse IDE (<http://www.eclipse.org>) (or similar).
* Code must be provided in annex and printed directly from Eclipse.
* Code that does not compile will be graded as 0.

All your code must be available on GitHub under the CS361 and Homework2 directories.

* Your homework must be well presented and have a cover page. 10 points will be reduced from your grade if you do not do have a cover page.
* The presentation of the hard copy of your homework assignment must contain your name(s).
* In case of problems with this homework, contact me by email [cscharff@pace.edu](mailto:cscharff@pace.edu).
* Grade: 100 points

**Question 1:**

Consider the following code. Each *draw* method has a number.

public class Circle{

public double center\_x, center\_y;

public double radius;

public void draw() {

// **(1)** method to draw circle on the screen

}

public void draw(Color color) {

// **(2)** method to draw circle on the screen with a

// given color

}

}

public class ColoredCircle extends Circle{

public int color;

public void draw() {

// **(3)** method to draw the colored circle

}

}

1. Explain polymorphism on the code above.

In the above code, ColoredCircle is the subclass of Circle and the both have the method draw().One is (1) and the other one is (2).But the have different behaviors when they are invoked.

1. c is of type Circle and d is of type ColoredCircle. Can we write d = c;? Why?

No,we can’t.Because ColoredCircle is the subclass of Circle and jvm can’t support that.

1. c is of type Circle and d is of type ColoredCir cle. Can we write c = d;? Why? What happens if we execute the code below? What method called *draw* is called? Why?

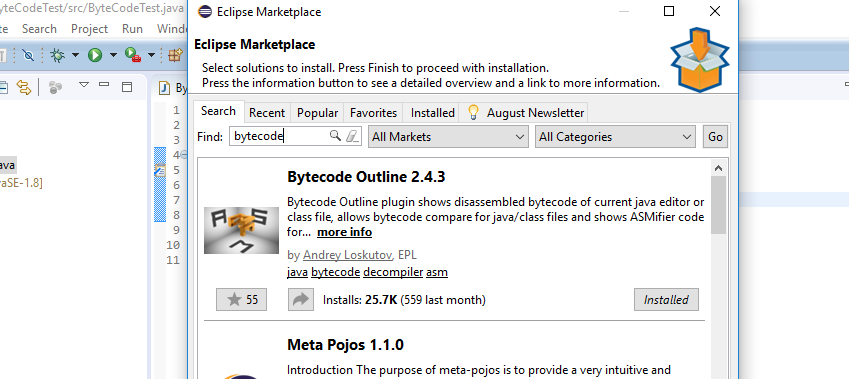
Yes,we can. Because ColoredCircle is the subclass of Circle.When we write c=d,the object c will have all the behaviors of ColoredCircle.So method (3) will be called.

c = d;

c.draw();

**Question 2:**

Install the following Eclipse Bytecode Outline plugin from: <http://asm.objectweb.org/eclipse/index.html> or from the Eclipse MarketPlace.



*[Dr. Scharff tested with the Neon version of Eclipse and with Eclipse Marketplace Byte Outline 2.4.3 plugin and it works! ]*

1. What Eclipse version are you using?

4.8.0.

1. What Java version are you using?

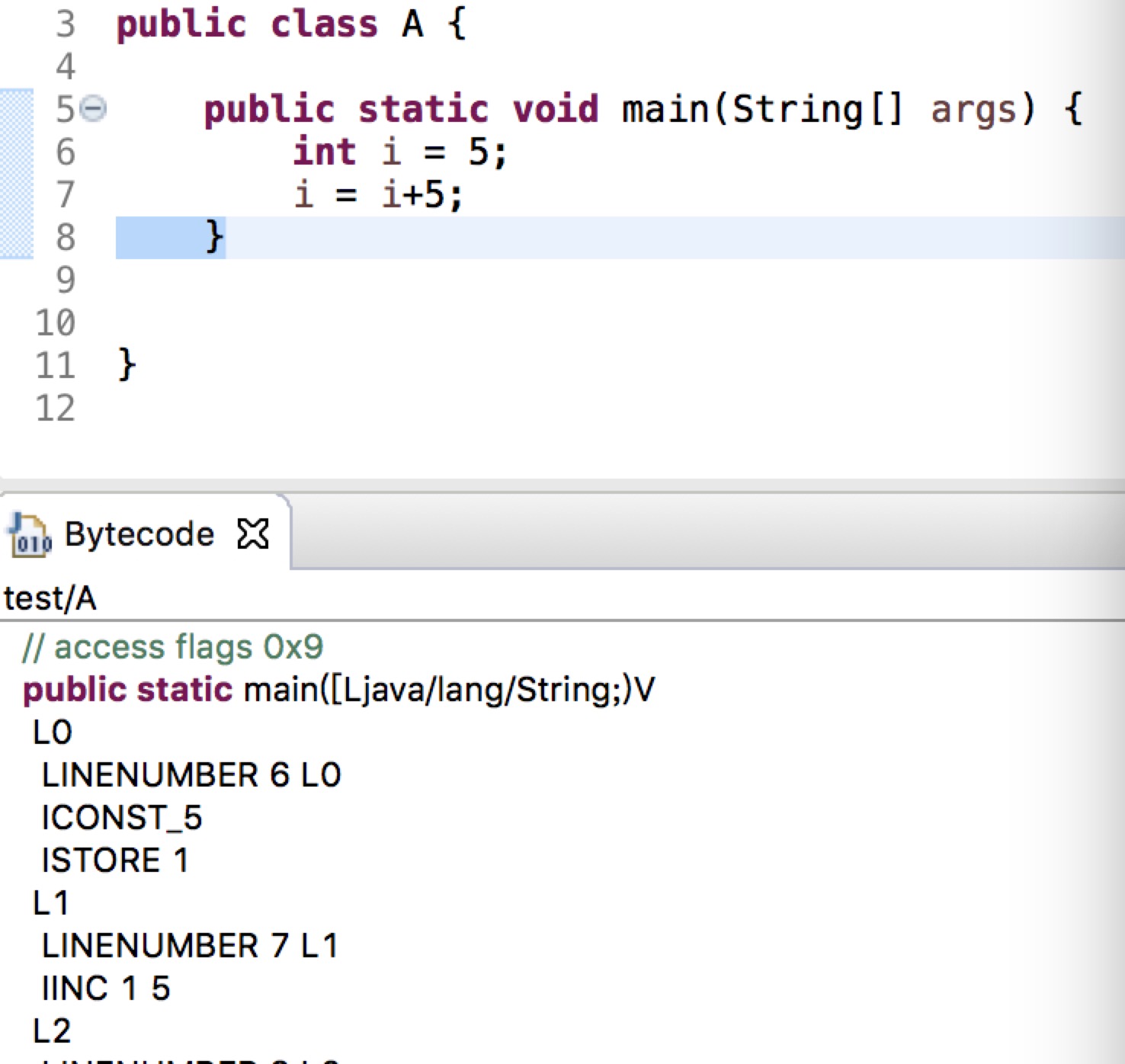
1.8.0\_172.

1. What is the Bytecode generated by the following statements?

int i = 5;

i = i+5;

Explain the syntax of the Bytecode. Provide a screenshot to support your work.



ICONST\_5:push the constant 5 onto the stack.

ISTORE 1:pop 5 off the stack and store it in local variable i.

IINC 1 5:increment local variable i by 5.

1. Compare the Bytecode generated by the 2 functions below and write down your conclusions.

Provide screenshots to support your work.

**public** **static** **int** sum\_for(**int** n) {

**int** i = 0, sum = 0;

**for** (i = 0; i <= n; i++) {

sum += i;

}

**return** sum;

}

**public** **static** **int** sum\_while(**int** n) {

**int** i = 0, sum = 0;

**while** (i <= n) {

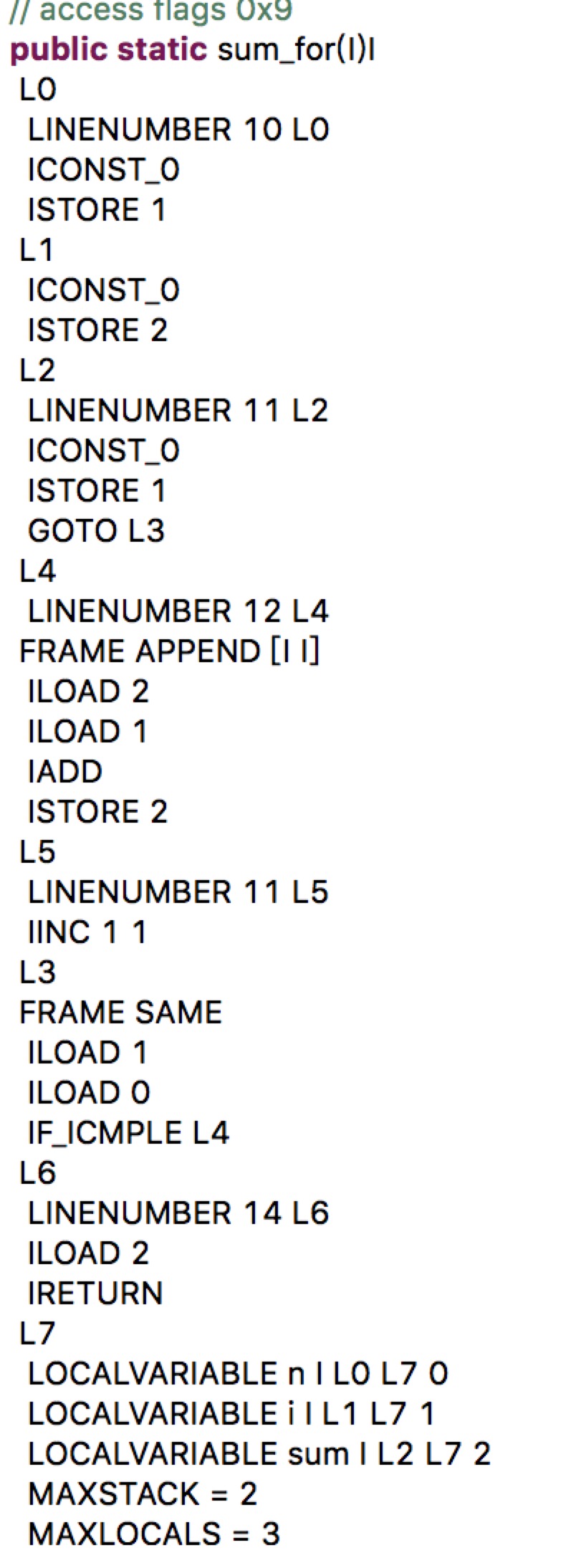
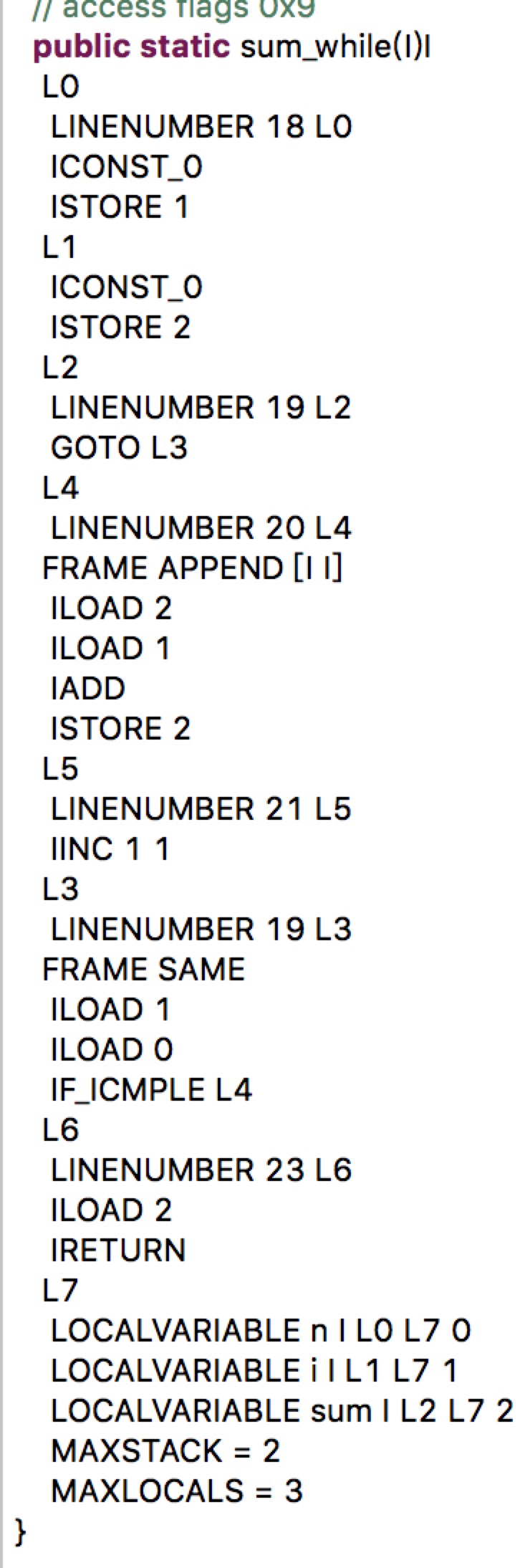
sum += i;

i++;

}

**return** sum;

}

In conclusion,the two functions have the same bytecode.

1. Write the factorial function (with the profile: public static fact(int n)) and describe the bytecode generated by this function.

**public** **static** **int** fact(**int** n) {

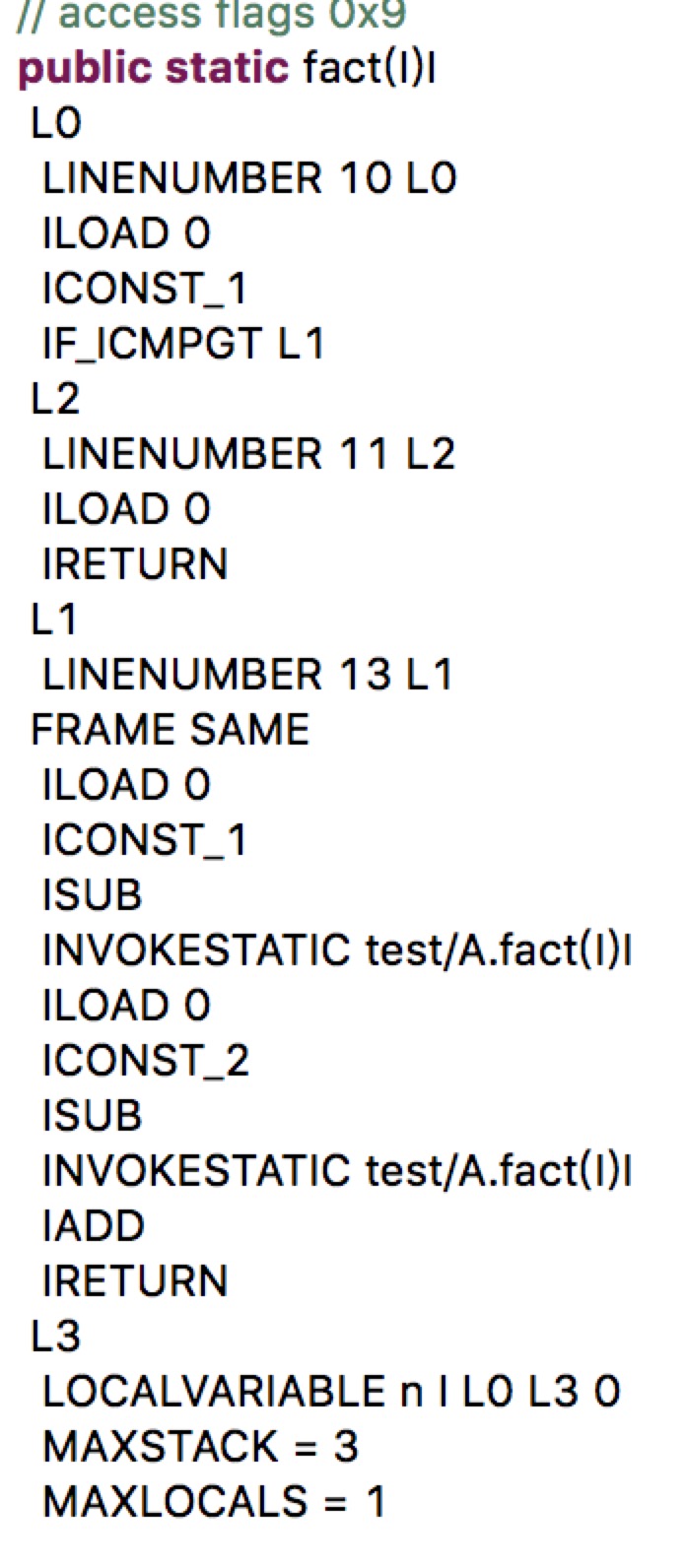
**if**(n<=1) {

**return** n;

}

**return** *fact*(n-1)+*fact*(n-2);

}



ILOAD 0:push int value held in the local variable n onto the operand stack.

ICONST\_1:push the int 1 onto the stack.

IF\_ICMPGT L1:pop the top two ints and compare them.jump to L1 if value2 is greater than value1.

ICONST\_1: push the int 1 onto the stack.

IRETURN:pop an int from the top of the stack and pushes it onto theoperand stack of the invoker.

ILOAD 0:push int value held in the local variable n onto the operand stack.

ICONST\_1:push the int 1 onto the stack.

ISUB: pop two ints off the operand stack, subtracts the top one from the second , and pushes the int resultback onto the stack.

INVOKESTATIC test/A.fact(I)I:invoke the class method fact.

ILOAD 0:push int value n held in the local variable n onto the operand stack.

ICONST\_2:push the int 2 onto the stack.

ISUB: pop two ints off the operand stack, subtracts the top one from the second , and pushes the int resultback onto the stack.

INVOKESTATIC test/A.fact(I)I:invoke the class method fact.

IADD: pop two integers from the operand stack, adds them, and pushes theinteger result back onto the stack.

IRETURN:pop an int from the top of the stack and pushes it onto theoperand stack of the invoker.

1. Choose a tail recursive function and describe the bytecode generated by this function. Compare with the code generated for a recursive function obtained in c).

What tail recursive function did you choose?

**public** **static** **int** fact(**int** n,**int** acc1,**int** acc2) {

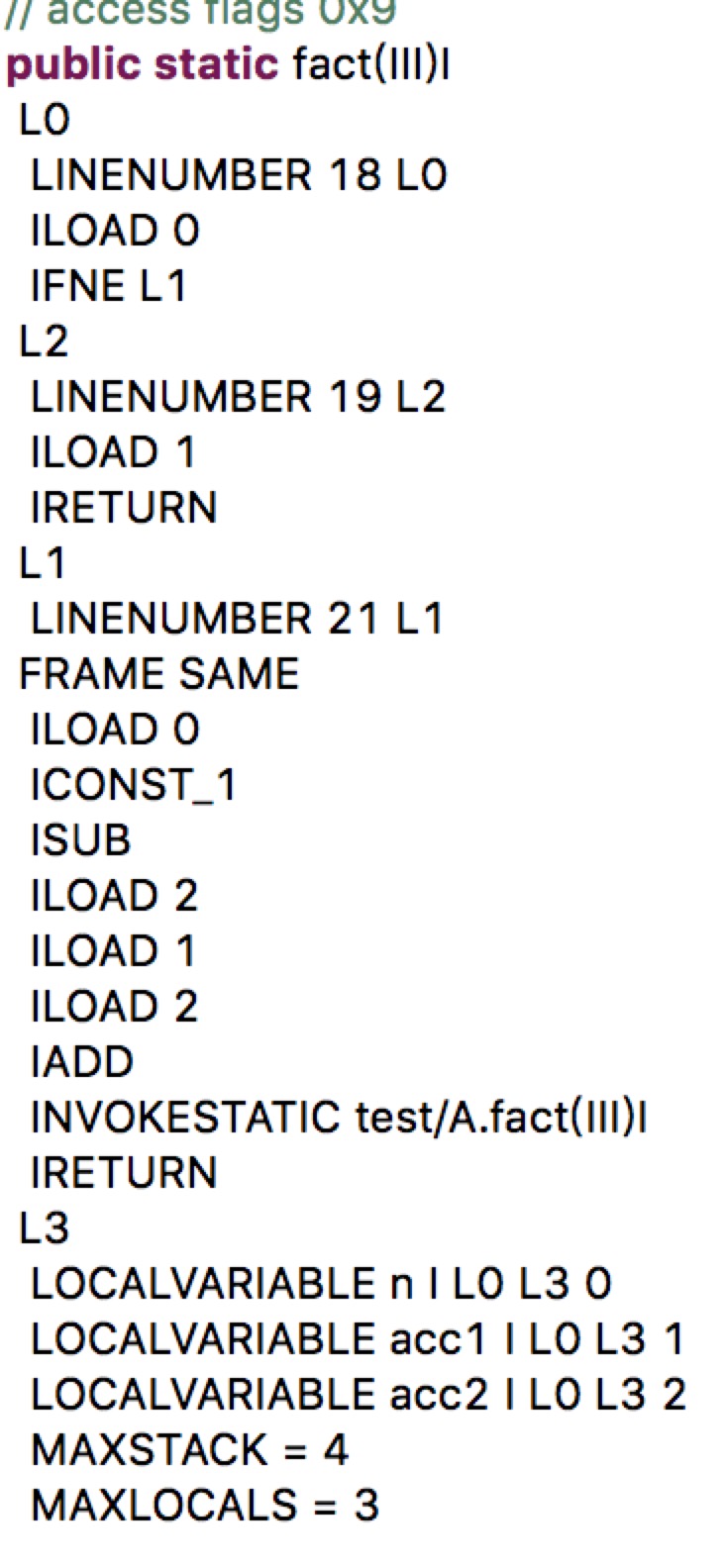
**if**(n==0) {

**return** acc1;

}

**return** *fact*(n-1, acc2, acc1+acc2);

}



ILOAD 0: push int value held in the local variable n onto the operand stack.

IFNE L1:pop the top int off the operand stack. If the int doesn’t equal zero jump to L1.

ILOAD 1: push int value held in the local variable I onto the operand stack.

IRETURN: pop an int from the top of the stack and pushes it onto theoperand stack of the invoker.

ILOAD 0: push int value held in the local variable n onto the operand stack.

ICONST\_1: push the int 1 onto the stack.

ISUB: pop two ints off the operand stack, subtracts the top one from the second , and pushes the int resultback onto the stack.

ILOAD 2: push int value held in the local variable acc2 onto the operand stack.

ILOAD 1: push int value held in the local variable acc1 onto the operand stack.

ILOAD 2: push int value held in the local variable acc2 onto the operand stack.

IADD: pop two integers from the operand stack, adds them, and pushes theinteger result back onto the stack.

INVOKESTATIC test/A.fact(III)I: invoke the class method fact.

IRETURN: pop an int from the top of the stack and pushes it onto theoperand stack of the invoker.

I will choose tail recursive for that will decrease the count of recursive.

**References**

* The Java Virtual Machine Specification <https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf> (Java 8 SE)
* Java Bytecode Basics <http://www.javaworld.com/javaworld/jw-09-1996/jw-09-bytecodes.html> (1996)
* <http://www.beyondjava.net/blog/java-programmers-guide-java-byte-code/> (2015)

**Question 3:**

1. Write a PROLOG program that describes the British family until nowadays. Kate, William and their children should be cited in the facts. Your program will start with the facts available in the slides (slide 31) and ends with Kate, William and their children. Draw a genealogy tree first.
2. Write a **rule** that describes the father predicate. *Father(X,Y)* means that *X* is the father of *Y*.

**Question 4:**

Write a **recursive** function *recPow* that computes 2n for n >= 0 in Java. The function will have the following profile:

public static int recPow(int n)

The function must consider all cases and be tested exhaustively. Show your testing!

**public** **static** **int** recPow(**int** n) {

**if**(0==n) {

**return** 1;

}

**return** 2\**recPow*(n-1);

}

**public** **static** **void** test() {

**for**(**int** i=0;i<10;++i) {

System.***out***.print((*recPow*(i)==Math.*pow*(2, i))+" ");

}

}

Output: true true true true true true true true true true

**Question 5:**

Write a **recursive** function merge that merges 2 arrays in Java. . The function will have the following profile:

public static void mergeSort(int[] a)

The function must be tested exhaustively. Show your testing!

If you use code online, you will need to cite your sources.

**public** **static** **void** mergeSort(**int**[] a) {

*Sort*(a, 0, a.length - 1);

}

**private** **static** **void** Sort(**int**[] a, **int** left, **int** right) {

**if** (left >= right) {

**return**;

}

**int** mid = (left + right) / 2;

*Sort*(a, left, mid);

*Sort*(a, mid + 1, right);

*merge*(a, left, mid, right);

}

**private** **static** **void** merge(**int**[] a, **int** left, **int** mid, **int** right) {

**int**[] tmp = **new** **int**[a.length];

**int** r1 = mid + 1;

**int** tIndex = left;

**int** cIndex = left;

**while** (left <= mid && r1 <= right) {

**if** (a[left] <= a[r1]) {

tmp[tIndex++] = a[left++];

} **else** {

tmp[tIndex++] = a[r1++];

}

}

**while** (left <= mid) {

tmp[tIndex++] = a[left++];

}

**while** (r1 <= right) {

tmp[tIndex++] = a[r1++];

}

**while** (cIndex <= right) {

a[cIndex] = tmp[cIndex];

cIndex++;

}

}

**public** **static** **void** test() {

**int**[] a = **new** **int**[] { 3, 8, 1, 10, 2, 7, 5, 3, 1, 9 };

*mergeSort*(a);

System.***out***.println(Arrays.*toString*(a));

}

Output: [1, 1, 2, 3, 3, 5, 7, 8, 9, 10]

**Question 6:**

Dijkstra's algorithm for gcd is the following:

gcd(m,n) = m if m = n

gcd(m-n, n) if m > n

gcd(m, n-m) if m < n

1. Is this definition well-formed? Explain.

No.It should be as follows:

gcd(m,n) =m if m = n

=gcd(m-n, n) if m > n

=gcd(m, n-m) if m < n

1. Is this definition well-defined? Explain.

Yes.Because it can work correctly.

1. Is this definition tail recursive? Explain.

Yes.Beacuse each step used the last step’s result.

1. Evaluate gcd(20,30) and show EACH step.

gcd(20,30)=gcd(20,10);

gcd(20,10)=gcd(10,10);

gcd(10,10)=10;

1. Implement gcd in Java with the following profile:

public static int gcd(int n,int m)

**public** **static** **int** gcd(**int** n,**int** m) {

**if**(m==n) {

**return** m;

}

**if**(m>n) {

**return** *gcd*(m-n, n);

}**else** {

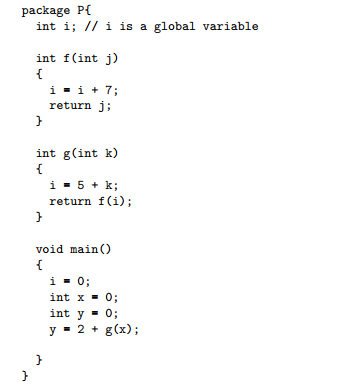
**return** *gcd*(m, n-m);

}

}

**Question 7:**

We consider the code below:

**

Draw the state of the memory during the execution of the code above using the following specifications:

* k is passed by value in g.
* j is passed by reference in f.
* You will use the drawing conventions seen in class.

Each step must be visible on the drawing.

What is the value of y at the end of the execution of the code?

i is assigned with 0 in main.

x is defined and assigned with 0 in main.

y is defined and assigned with 0 in main.

k is passed by value in g.

i is assigned with 5 in main.

J is passed by value in f.

i is assigned with 12 in f.

y is assigned with 7 in main.

So y will be 7 at the end of the execution of the code.