**Matthew Sokolovsky**

**CS 361**

**Homework 2**

**Programming Languages Principles and Implementation**

**Question 1: History of programming languages**

Put the following programming languageson [sic] a chronological timeline. The year must be provided. **In addition,** indicate the name of the designer of the programming language, where it was created (company, national lab, higher education institution etc.), and the country.

* 1957: Fortran, designed by John Backus at IBM in the United States
* 1958: Lisp, designed by John McCarthy at MIT in the United States
* 1959: Cobol, designed by Grace Hopper at the Conference on Data Systems Languages in the United States
* 1970: PASCAL, designed by Niklaus Wirth at ETH Zurichin Switzerland
* 1972: Prolog, designed by Alain Colmerauer and Robert Kowalski at the University of Aix-Marseille in France and the University of Edinburgh in the United Kingdom
* 1972: C, designed by Dennis Richie at Bell Labs in the United States
* 1980: ADA, designed by Jean Ichbiah at the Department of Defense in the United States
* 1983: C++, designed by Bjarne Stoustrup at Bell Labs in the United States
* 1985: EIFFEL, designed by Bertrand Meyer at Interactive Software Engineering (now known as Eiffel Software) in California, US.
* 1986: ISETL, designed by Ed Dubinsky and Gary Levin at Clarkson University in the United States
* 1987: Perl, designed by Larry Wall at Unisys in the United States
* 1988: Standard ML, designed by Robin Milner at the University of Edinburgh in the United Kingdom
* 1991: Python, designed by Guido Van Rossum at Centrum Wiskunde & Informatica in the Netherlands
* 1995: Java, designed by James Gosling at Sun Microsystems in the United States
* 1995: Ruby, designed by Yukihiro Matsumoto in Japan
* 2016: Kotlin, designed by the company JetBrains in the Czech Republic

**Question 2:**

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

}

}

For Question 2, I copied the above code into Eclipse, creating a Circle class and a ColoredCircle class. I also created another class with a main method to do tests. I changed the parameter of the second draw method to an int (since the Color class was undefined. I also made each method print out a different string, so when I called a draw method I could tell which one I had called.

**public** **class** Circle {

**public** **double** center\_x, center\_y;

**public** **double** radius;

**public** **void** draw() {

System.***out***.println("Method A");

}

**public** **void** draw(**int** color) {

System.***out***.println("Method B");

}

}

**public** **class** ColoredCircle **extends** Circle {

**public** **int** color;

**public** **void** draw() {

System.***out***.println("Method C");

}

}

**public** **class** HW2\_Main {

**public** **static** **void** main(String[] args) {

Circle c = **new** Circle();

ColoredCircle d = **new** ColoredCircle();

}

}

1. Explain polymorphism on the code above.

Objects of the ColoredCircle class inherit all of the variables of the Circle class (*center\_x*, *center\_y* and *radius*), as well as the methods of the circle class. However, since both classes have a method with the same type, name and parameters (*public void draw()*), the ColoredCircle class will override the parent version of the method. This means that if you have a ColoredCircle object *d* and you call *d.draw()* it will do the ColoredCircle version of the method rather than the Circle version.

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

No, it causes an error. An object of the parent class can’t be assigned to an object of a child class. I don’t know why Java doesn’t allow this, but I have tested the above statement in Eclipse and it causes an error.

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

c = d;

c.draw();

I tested the above code and found that it *is* possible to do c = d;. Although child\_object = parent object; doesn’t work, parent\_object = child\_object; does. The statement c.draw(); calls the Colored Circle version of draw() rather than the Circle version.

**Question 3:**

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

1. What Eclipse version are you using?

I’m using Eclipse Oxygen.

1. What Java version are you using?

I’m using Java 8.

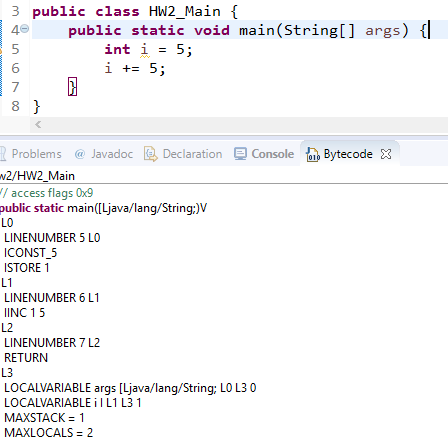
1. What is the Bytecode generated by the following statements?

int i = 5;

i = i+5;

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

The screenshot below shows the bytecode compiled from those two lines.



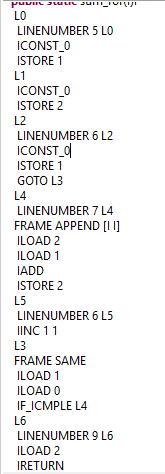
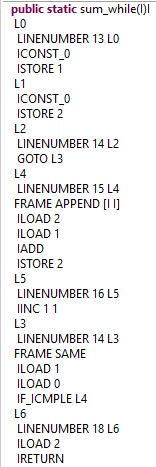
The LINENUMBER statement says that the following code (until the next LINENUMBER statement) takes place on the specified line. ICONST\_5 fetches a constant equal to 5. ISTORE stores the fetched value in the memory. The variable i is identified by the number 1.

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

Provide screenshots to support your work.

***/\*Professor Scharff, you forgot to put spaces between publicstaticint. Because of your mistakes, your code receives a grade of 0.\*/***

**publicstaticint** sum\_for(**int** n) {



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

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

sum += i;

}

**return** sum;

}

**publicstaticint** sum\_while(**int** n) {

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

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

sum += i;

i++;

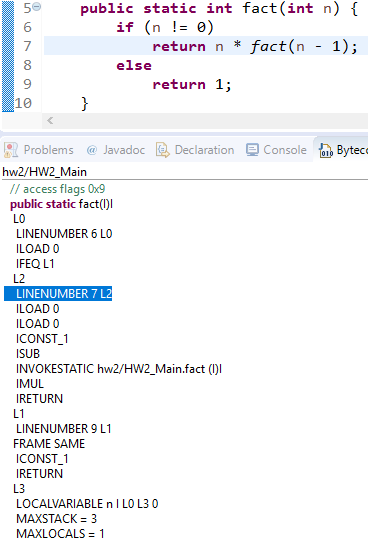
}

**return** sum;

}

The two bytecodes are mostly the same. The only differences lie in the LINENUMBER lines and the fact that ICONST\_0 and ISTORE 1 are done twice in the for loop (this is something that could have been avoided by not having *int i = 0* at the top).

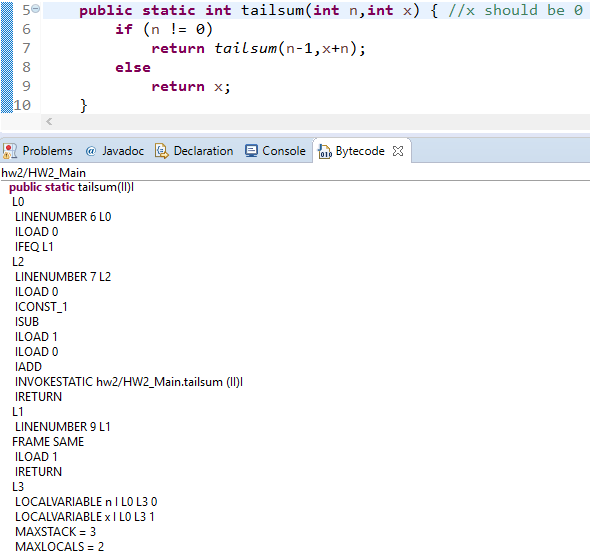
1. Write the factorial function (with the profile: public static fact(int n)) and describe the bytecode generated by this function.



ILOAD 0 gets the value of n. Rather than checking if n != 0 as my code says, it checks the opposite. If the opposite (n == 0) is true, it ignores L2 (where *return n \* fact(n – 1)* is) and jumps down to L1, where it returns 1. When it goes to L2, it loads n twice, which means that it puts n on the stack twice. ICONST\_1 puts 1 on the stack, and ISUB subtracts the 1 from the upper n and replaces them with n – 1. INVOKESTATIC triggers the recursive step fact(n – 1). IMUL multiplies the the other n and the result of fact(n – 1) and IRETURN returns the thing on the top of the stack if it’s an integer.

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).

I made a tail recursion based on the code from part d; the code and bytecode are below.



Here’s what the bytecode represents:

L0 if (n == 0) go to L1, else go to L2

L2 return tailsum(n-1,x+n)

L1 return x

**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 4:**

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.

P(Edward VII, George V).

P(Victoria, Edward VII).

P(Alexandra, George V).

P(George VI, Elizabeth II).

P(George V, George VI).

P(Elizabeth II, Charles)

P(Charles,William)

P(William,George)

P(Kate,George)

P(William,Charlotte)

P(Kate,Charlotte)

G(x,y):-P(x,z),P(z,y).

1. Write a **rule** that describes the father predicate. *Father(X,Y)* means that *X* is the father of *Y*.

Father(X,Y):-Male(X),P(X,Y)

//For this to work, I’d also have to define people as Male

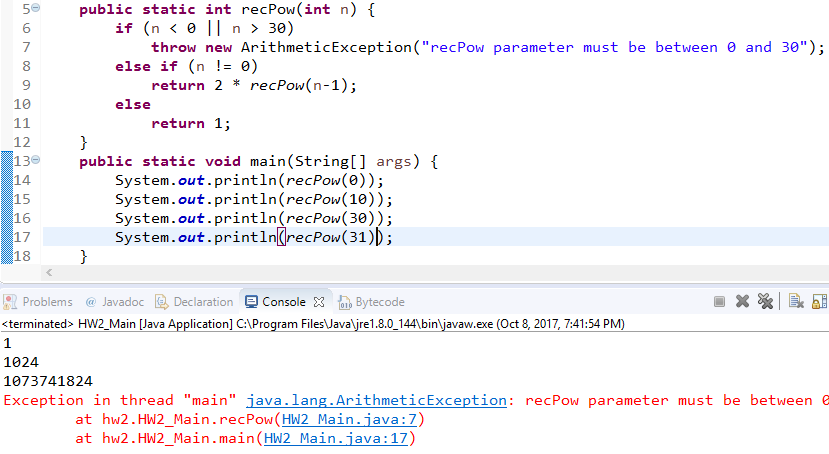
**Question 5:**

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!

Since an int can only be 231 - 1 at most, the highest n can be is 30. In the screenshot below I show my code and my testing.



**Question 6:**

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

public static int[] mergeSort(int[] a, int[] b)

You will use the split function of slide 18 (odd and even positions).

The function must be tested exhaustively. Show your testing!

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

Here’s my mergeSort program.

**public** **static** **int**[] mergeSort(**int**[] a, **int**[] b) {

**if** (a.length > 1) { //If a still has more than one element, it is split

**int**[] a\_left = **new** **int**[(a.length / 2) + (a.length % 2)]; //Rounded up

**int**[] a\_right = **new** **int**[a.length / 2]; //Rounded down

**int** index\_a = 0;

**for** (**int** i = 0; i < a\_left.length; i++, index\_a++)

a\_left[i] = a[index\_a]; //Puts elements into the left array

**for** (**int** i = 0; i < a\_right.length; i++, index\_a++)

a\_right[i] = a[index\_a]; //Puts elements into the right array

a = *mergeSort*(a\_left,a\_right); //Then it recursively sorts a

}

**if** (b.length > 1) { //Then it does the same with b

**int**[] b\_left = **new** **int**[(b.length / 2) + (b.length % 2)];

**int**[] b\_right = **new** **int**[b.length / 2];

**int** index\_b = 0;

**for** (**int** i = 0; i < b\_left.length; i++, index\_b++)

b\_left[i] = b[index\_b];

**for** (**int** i = 0; i < b\_right.length; i++, index\_b++)

b\_right[i] = b[index\_b];

b = *mergeSort*(b\_left,b\_right);

}

**int** index\_a = 0, index\_b = 0;

**int**[] c = **new** **int**[a.length + b.length]; //c is the sorted array we will return

**for** (**int** i = 0; i < c.length; i++) { //This loop sorts elements from a and b

**if** (index\_a < a.length && index\_b < b.length) {

**if** (a[index\_a] >= b[index\_b]) {

c[i] = a[index\_a]; //and puts them into c

index\_a++;

}

**else** {

c[i] = b[index\_b];

index\_b++;

}

}

**else** **if** (index\_a < a.length) {

c[i] = a[index\_a];

index\_a++;

}

**else** {

c[i] = b[index\_b];

index\_b++;

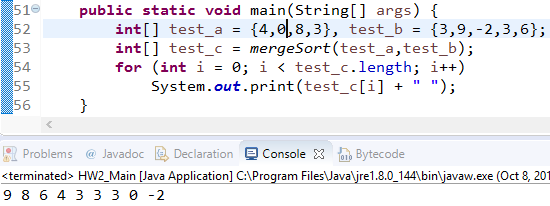
}

}

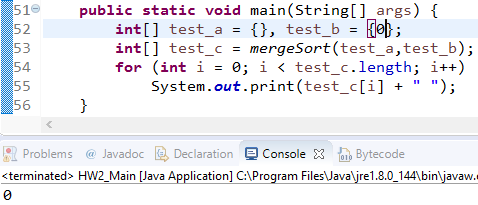
**return** c;

}

It’s not the most beautiful recursive function, but it works. I put screenshots of my tests on the next page.



As you can see, it sorts correctly (I made it sort from highest to lowest), and it can handle arrays with odd length, nonpositive numbers, and more than one of the same number. Here’s another test.



This test shows that it can handle arrays that begin completely empty or with only one element.