**Homework 2**

**Nick Krawczeniuk**

**worked with John Mulcahy**

**Programming Languages and Implementations  
Dr. Scharff****10/9/17**

**Homework 2**

**Programming Languages Principles and Implementation**

**Instructions:**

* Due date: 10/9 (No late homework will be accepted. The solution of the homework will be posted on 10/9 after class. The midterm is on 10/11.)
* 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](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: History of programming languages**

Put the following programming languages on 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.

* **Fortran –** 1954 (first draft completed),John W. Backus, IBM, United States
* **Lisp –** 1958, John McCarthy, MIT, United States
* **Cobol –** 1959, Howard Bromberg, Howard Discount, Vernon Reeves, Jean E. Sammet, William Selden, Gertrude Tierney**,** CODASYL, United States
* **ISETL** – 1969 (as SETL), Jacob T. Schwartz, NYU Courant, United States
* **PASCAL** – 1970, Niklaus Wirth, United States
* **Prolog –** 1972, Alain Colmerauer and Philippe Roussel, France
* **C** – 1972, Dennis Ritchie, AT&T Bell Labs, United States
* **SML –** 1973 as ML, SML standard in 1990, Robin Milner, University of Edinburgh, United Kingdom
* **ADA** – 1980, Jean Ichbiah, CII Honeywell Bull/Department of Defense, United States
* **C++** – 1983, Bjarne Stroustrup, AT&T Bell Labs, United States
* **EIFFEL –** 1986, Bertrand Meyer, Eiffel Software, France
* **Perl** – 1987, Larry Wall, Unisys, United States
* **Python** – 1991, Guido van Rossum, CWI, Netherlands
* **Java –** 1995, James Gosling, Mike Sheridan, and Patrick Naughton, Sun Microsystems (now part of Oracle), United States
* **Ruby –** 1995, Yukihiro Matsumoto, Japan
* **Kotlin –** 2011, JetBrains team, 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

}

}

1. **Explain polymorphism on the code above.**
   1. The ColoredCircle class extends Circle. It draws a circle (sourcing from the parent Circle class) and then uses another drawing method to color the circle.

1. **c is of type Circle and d is of type ColoredCircle. Can we write d = c;? Why?**
   1. Yes, because ColoredCircle is a type of Circle.
2. **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?** 
   1. ColoredCircle is a Circle with a color attribute but a Circle does not necessarily have color so c = d is not correct. If you execute the code below, it won't work. It's not sure which draw method to call so it fails.

c = d;

c.draw();

**Question 3:**

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?

**4.5.2**

1. What Java version are you using?

**Version 8, update 144 (JavaSE-1.8)**

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.



**init:**

ALOAD 0 pushes an object reference in local zero to the stack, INVOKESPECIAL invokes the initialization method <init>

**main:**

L0 (associated with Line 6):

ICONST\_5 pushes int 5 onto i in the stack

ISTORE 1 stores it

L1 (associated with Line 7):

IINC 1; i 5 increments the number in i by 5

L2 (associated with Line 9):

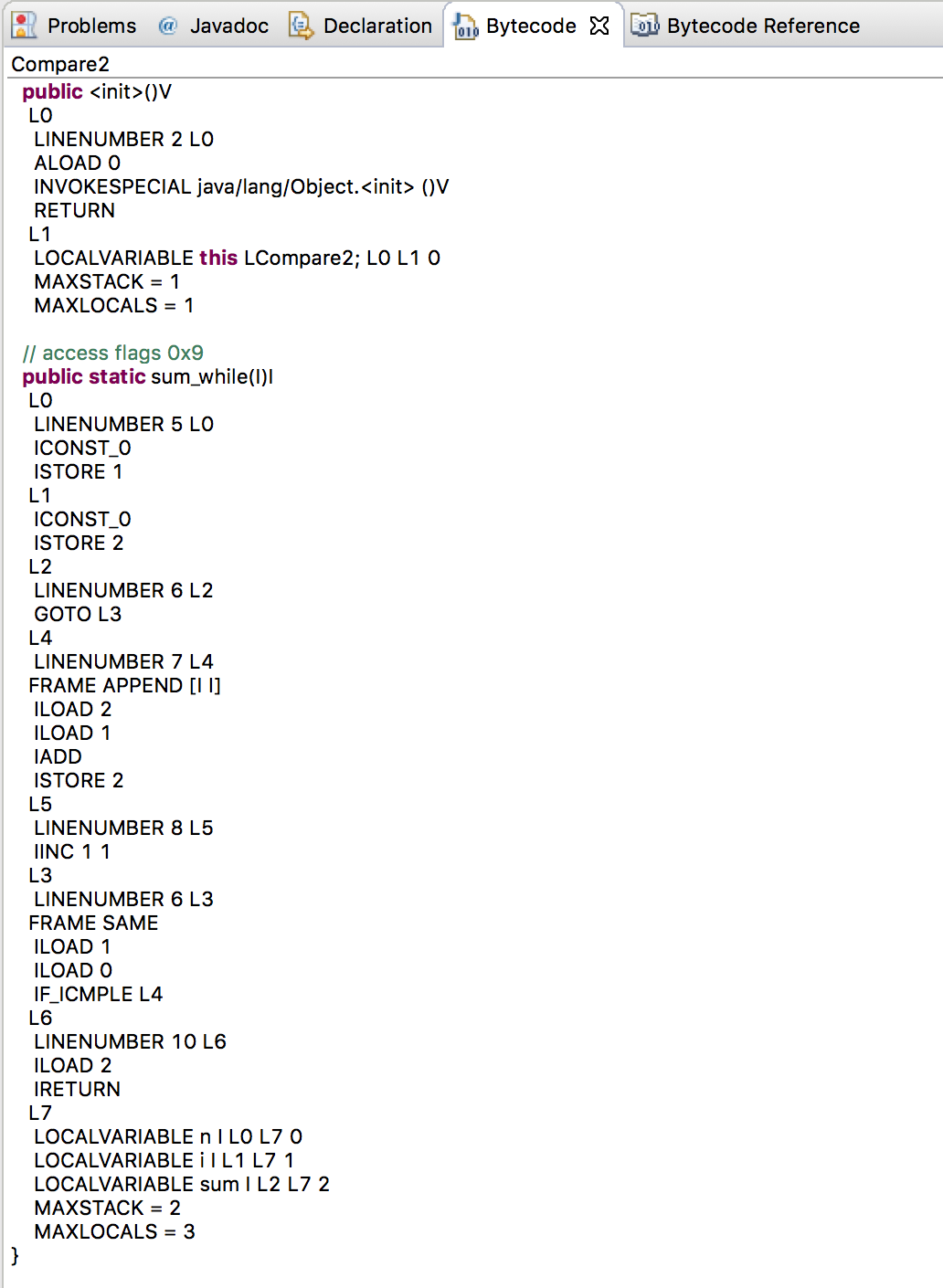
RETURN returns nothing

L3 (associated with the main method):

Notes the args[] String array and the other local int variables as well as the maximum number of items and variables that have been used.

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;

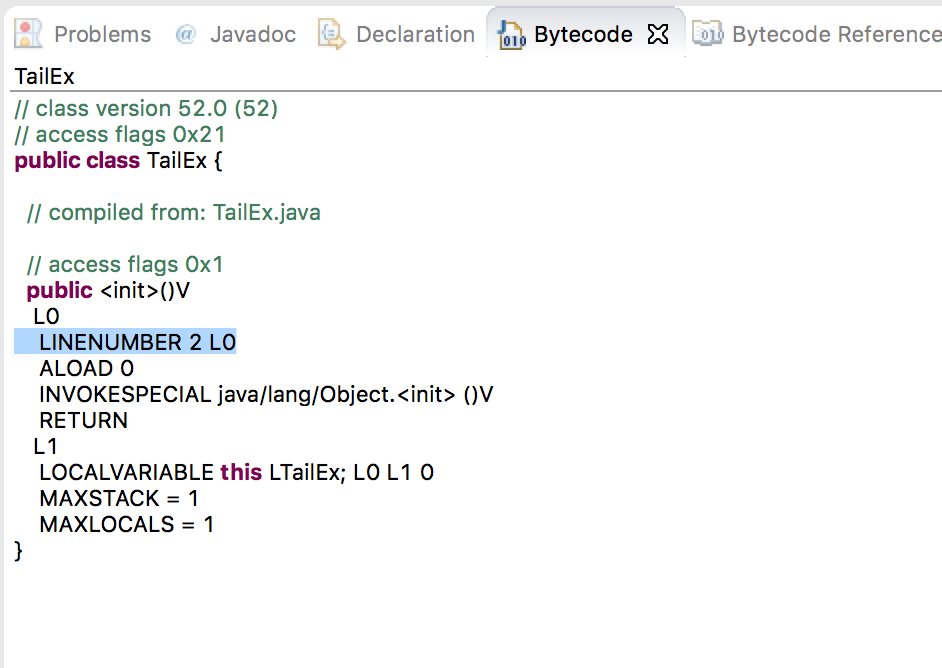
}

Both initialize the same way with INVOKESPECIAL etc. Compare1 in L2 pushes a constant 0 to the stack and stores 1 then goto's to the address. Compare2 simply goto's directly to the address rather than pushing and storing.

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

The bytecode shows a local reference being loaded into local variable zero. Then it invokes an object and puts the result on the stack. After that, it loads a value into the local variable and loads zero. It pushes zero if the two variables are the same, 1 if one value is greater than the other, and -1 otherwise. If the value isn't zero, it branches off to instruction at branchoffset. It continues down this path and loads values into the stack eventually calling upon itself in INVOKESTATIC facto.fact (J)J.

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).



**public** **class** TailEx {

**public** **void** printName(**int** level){

**if**( level <= 0 )

**return**;

System.***out***.println("Smith");

printName(--level);

}

}

<https://stackoverflow.com/questions/11592796/tail-recursion-in-java>

The bytecode is different in that it does not call on itself such as INVOKESTATIC in the factorial example above. The tail recursive function also does not continue once it is finished with its computations.

**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.

:- initialization(main).

Male(Edward VII).

Male(George V).

Male(George VI).

Male(Charles).

Male(William).

Male(Phillip).

Male(George VIII).

Male(Harry).

P(Victoria, Edward VII).

P(Edward VII, George V).

P(Alexandra, George V).

P(George V, George VI).

P(George VI, Elizabeth II).

P(Elizabeth II, Charles).

P(Phillip, Charles).

P(Charles, William).

P(Charles, Harry).

P(Diana, William).

P(William, George VIII).

P(Kate, George VIII).

P(William, Charlotte).

P(Kate, Charlotte).

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

Father(x,y) :- P(x,y), Male(x).

**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!

**import** java.lang.Math;

**import** java.util.Scanner;

**public** **class** Assign2 {

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

Scanner user\_input = **new** Scanner(System.***in***);

System.***out***.println("Enter a value for n between 0 and 30:");

**int** n = user\_input.nextInt();

**int** answer = *recPow*(n);

**if**(n>=0 && n<=30)

System.***out***.println(answer);

}

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

**int** ans=1;

**if** (n< 0 || n>30){

System.***out***.println("Number must be greater than 0 and less than 30");

**return** 0;

}

**else**{

**if**(n==0)

**return** ans;

**else**

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

}

}

**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.

**public** **class** mergeSort {

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

**int**[] array = **new** **int**[]{1, 6, -23, 9, 10, -7, 12, 32, 56};

**int**[] array1 = **new** **int**[]{2, 4, -23, 9, 10, 78, 98, 87, 1, 3, 102};

**int**[] array2 = **new** **int**[]{12, -89, 9, 10, 15, 56, 8, 6, 87, 12};

**int**[] array3 = **new** **int**[]{4, 12, 25, 9, 10, 4, 1, 0, -45};

**int**[] array4 = **new** **int**[]{9, 55, -89, 9};

**int**[] array5 = **new** **int**[]{54, 66};

*sortArray*(array);

*printArray*(array);

System.***out***.print("\n");

*sortArray*(array1);

*printArray*(array1);

System.***out***.print("\n");

*sortArray*(array2);

*printArray*(array2);

System.***out***.print("\n");

*sortArray*(array3);

*printArray*(array3);

System.***out***.print("\n");

*sortArray*(array4);

*printArray*(array4);

System.***out***.print("\n");

*sortArray*(array5);

*printArray*(array5);

System.***out***.print("\n");

}

**public** **static** **void** mergeSort(**int**[] array, **int** splitlow, **int** splithigh){

**if**(splitlow < splithigh){

**int** middle = (splitlow + splithigh) / 2;

*mergeSort*(array, splitlow, middle);

*mergeSort*(array, middle+1, splithigh);

*merge*(array, splitlow, middle, splithigh);

}

**else**

**return**;

}

**public** **static** **void** sortArray(**int**[] array){

*mergeSort*(array, 0, array.length - 1);

}

**public** **static** **void** printArray(**int** [] array){

**for**(**int** i : array)

System.***out***.printf("%d ", i);

}

**public** **static** **void** merge(**int** array[], **int** splitlow, **int** middle, **int** splithigh){

**int** leftsize = middle - splitlow +1;

**int** rightsize = splithigh - middle;

**int**[] left = **new** **int**[leftsize + 1];

**int**[] right = **new** **int**[rightsize + 1];

left[leftsize] = Integer.***MAX\_VALUE***;

right[rightsize] = Integer.***MAX\_VALUE***;

**for**(**int** i = 0; i < leftsize; i++)

left[i] = array[splitlow +i];

**for**(**int** i = 0; i < rightsize; i++)

right[i] = array[i+middle+1];

**int** temp = 0;

**int** temp1 = 0;

**for**(**int** i = splitlow; i <= splithigh; ++i){

**if**(left[temp] <= right[temp1]){

array[i] = left[temp];

temp++;

}

**else** {

array[i] = right[temp1];

temp1++;

}

}

}

}

<http://www.softwareandfinance.com/Java/MergeSort_Recursive.html>  
<http://andreinc.net/2010/12/22/the-merge-sort-algorithm-implementation-in-java/>  
<http://www.vogella.com/tutorials/JavaAlgorithmsMergesort/article.html>  
<http://www.softwareandfinance.com/Java/MergeSort_Recursive.html>