**Pace University**

**Homework 2**

**Programming Languages Principles and Implementation**

**October 9,2017**

**CS361**

**Angie Ramirez (**[**ar15449n@pace.edu**](mailto:ar15449n@pace.edu)**)**

**Christopher Thomas([Ct61632n@pace.edu](mailto:Ct61632n@pace.edu))**

**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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Designer, company** | **Year** | **Country** |
| Fortran | John W. Backus at IBM | 1957 | USA |
| Cobol | Grace Hopper of the US Navy | 1959 | USA |
| SML | Robin Milner & others at the University of Edinburgh | 1973 | Edinburgh, Scotland, UK |
| Prolog | Alain Colmerauer | 1972 | University of Aix-Marseille, France |
| C | Dennis Ritchie, ATT Bell Laboratories | 1972 | USA |
| PASCAL | Niklaus Wirth, Kathleen Jensen | 1970 | USA |
| EIFFEL | Bertrand Meyer | 1986 | USA |
| C++ | Bjarne Stroustrup, ATT Bell Laboratories | 1983 | USA |
| Java | James Gosling at Sun Microsystems | 1995 | USA |
| Ruby | Yukihiro Matsumoto | 1995 | Japan |
| Python | Guido van Rossum, Centrum Wiskunde & Informatica | 1991 | Netherlands |
| ADA | JeanIchbiah at CII Honeywell Bull | 1980 | France |
| ISETL | Jacob T. Schwartz, NYU | 1969 | USA |
| Lisp | John McCarthy, MIT | 1958 | USA |
| Perl | Larry Wall | 1987 | USA |
| Kotlin | JetBrains | 2011 | Russia |
|  |  |  |  |
|  |  |  |  |

**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.

Polymorphism in the code above allows the ColoredCircle class to refer to the Circle class for its methods so they do not have to be rewritten. So Circle is the superclass and the subclass ColoredCircle extends it.

1. c is of type Circle and d is of type ColoredCircle. Can we write d = c;? Why? Yes, because the program would just print a 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?

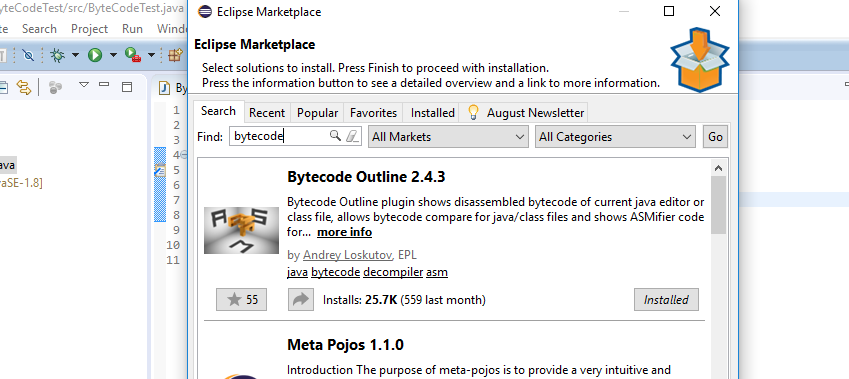
Yes you can write c =d because you are allowed to overwrite when using Polymorphism. The method draw from the subclass ColoredCircle is called, because c is being overwritten to equal d.

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?

Mars Release (4.5.0)

1. What Java version are you using?

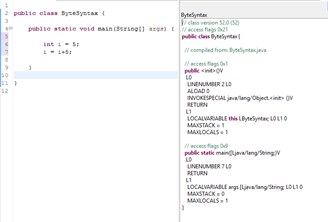
Version 8

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.



public <init> - Initializes

L0

LINENUMBER 2 L0 - Looks at line number 2

ALOAD 0 - Loads a reference onto the stack from local variable 0

INVOKESPECIAL - This method acts upon the object objectref and places the result on the stack.

RETURN - Returns void from a method

L1

LOCALVARIABLE thisLByteSyntaxL0L10 -

MAXSTACK = 1 - Max amount of stacks

MAXLOCALS = 1 Max amount of local variables.

public static main([Ljava/lang/String;)V

L0

LINENUMBER 7 L0

RETURN

L1

LOCALVARIABLE args [Ljava/lang/String; L0 L1 0

MAXSTACK = 0

MAXLOCALS = 1

}

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) {

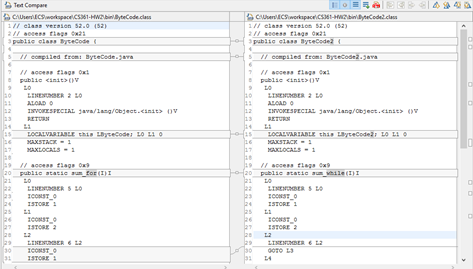
sum += i;

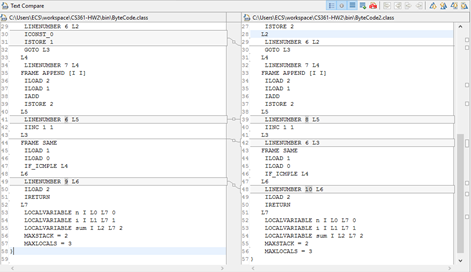
i++;

}

**return** sum;

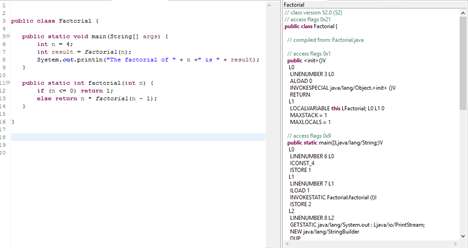
}

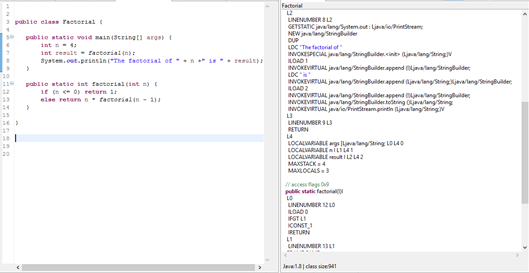


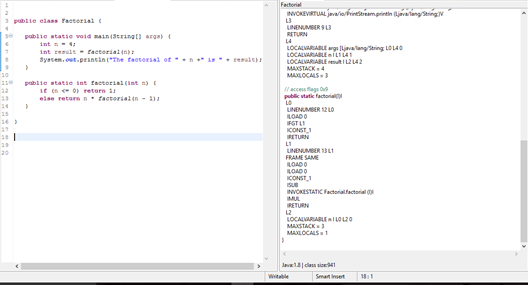


There are few differences between the two byte codes. The bytecode shown for class ByteCode shows the stack for a for loop, while the bytecode for the second class, ByteCode2, shows a stack for a while loop. The first class contains instructions at L2 called ICONST\_0 and ISTORE\_1. These instructions push the number 0 with type long onto the stack and stores a long value in local variable 1 respectively. The second class at L2 uses instruction GOTO L3, which goes to the instruction at the branchoffset.

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

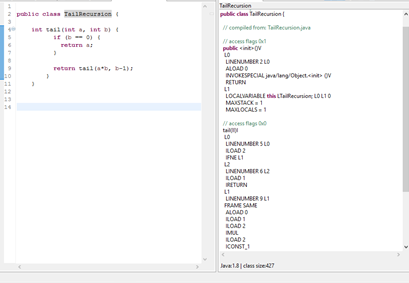


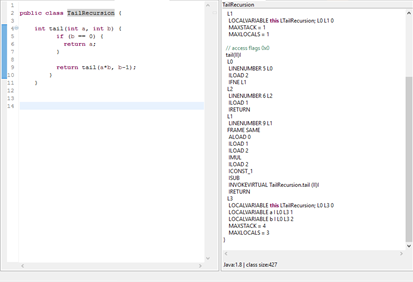




The bytecode generated shows a loop that goes back in the stack to evaluate if the numbers are less than or equal to 0. If it is not, it is deducted by 1 and returned to the stack until the value is 0.

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).





The bytecode corresponds to a loop.

**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(Elizabeth II, Charles). Male(Edward VII).

P(Victoria, Edward VII). P(Charles, Prince William). Male(George V).

P(Alexandra, George V). P(Charles, Prince Harry). Male(George VI).

P(George VI, Elizabeth II). P(Prince William,Prince George of Cambridge). Male(Prince Charles).

P(George V, George VI). P(Prince William,Charlotte). Male(Prince William).

G(x,y):-P(x,z),P(z ,y). Male(Prince Harry).

Male(x). Male(Prince Gorge of Cambridge).

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!

**public** **class** testre {

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

**if**(n == 0)

**return** 1;

**else**

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

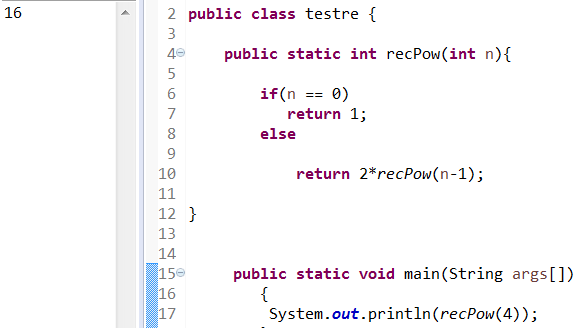
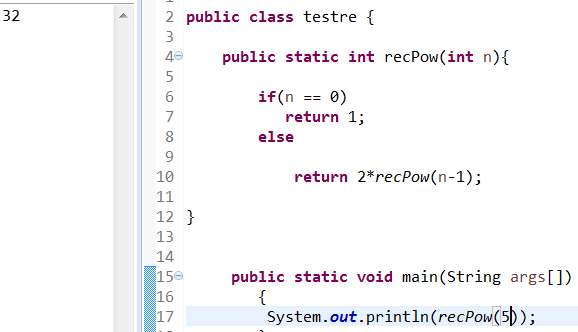
}

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

{

System.***out***.println(*recPow*(1));

}

} 

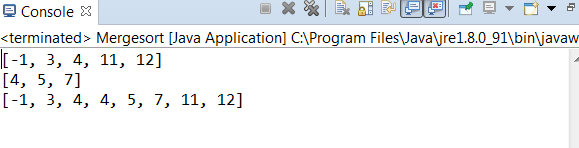
**Question 6:**

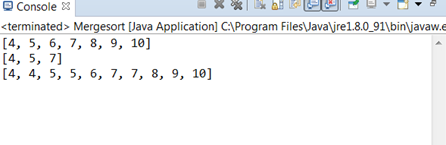
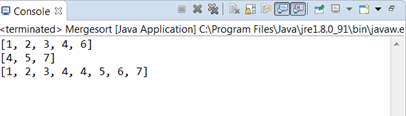
Write a **recursive** function merge that merges 2 arrays in Java. . The function will have the following profile:

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.



**import** java.util.\*;

**import** java.util.Arrays;

**public** **class** Mergesort {

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

**int**[] answer = **new** **int**[a.length + b.length];

**int** i = 0, h = 0, s = 0;

**while** (i < a.length && h < b.length)

{**if** (a[i] < b[h])

{ answer[s] = a[i];

i++;

}

**else**

{

answer[s] = b[h];

h++ }

s++;

}

**while** (i < a.length)

{

answer[s] = a[i];

i++;

s++;

}

**while** (h < b.length)

{

answer[s] = b[h];

h++;

s++;

}

**return** (answer);

}

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

{

**int**[] a={4,5,6,7,8,9,10};

**int**[] b={4,5,7}; System.***out***.println(Arrays.*toString*(a)); System.***out***.println(Arrays.*toString*(b)); System.***out***.println(Arrays.*toString*(*mergeSort*(a,b)));

}

}