**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>) (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
  + 1957
* Cobol
* SML
* Prolog
* EIFFEL
* C
* PASCAL
* C++
* Java
  + 1996
* Ruby
  + 1993
* Python
  + 1991
* ADA
* ISETL
* Lisp
* Perl
* Kotlin

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

We cannot write d=c because ColoredCircle is a Circle, but a Circle is not a ColoredCircle. Inheritance only goes up and cannot go down into the class. The only assignment that would work is c=d because then you would be saying that a ColoredCirlce is a Circle since Circle and its methods are inherited by the ColoredCircle class.

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?

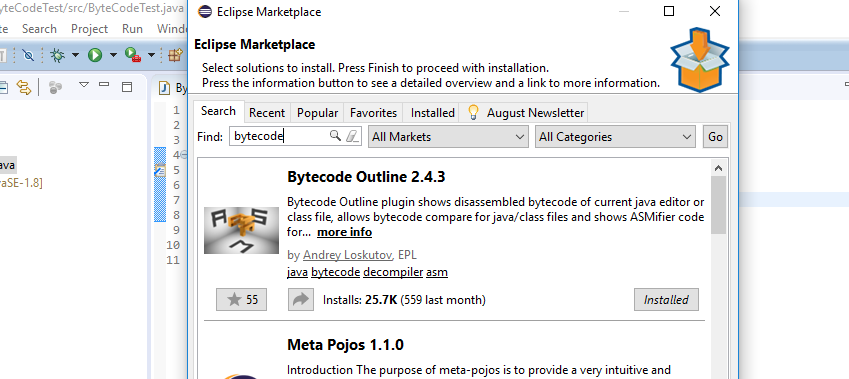
c = d;

c.draw();

Yes you can write c = d because ColoredCirlce inherits everything from Circle. If you run this line of code then the address for c will now be pointing to a d object and that object has it’s own draw method which is part of the Circle class. The names are only the same because of method overriding, but since the variable is now pointing to a Circle object, then the method called would be coming from the circle object and not the ColoredCircle object.

**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?
2. What Java version are you using?
3. 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 syntax above is fairly simple. The first block L0 reads linenumber 28 and then pushes an integer 5 onto the stack. Then the istore command pops the integer off of the stack and stores it into a local variable. The second block simply increments the value in the previous variable by 5.

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

These two blocks of code are completely the same except for the fact that i gets assigned 0 twice. This can be seen in the highlighted code below. The iconst\_0 pushes zero onto the stack and then istore 1:i pops it off of the stack and then saves it to the variable. This is double the work since the “i” was already assigned in the previous code.

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;

}

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

The byte code that is generated from this is pretty straight forward except for the recursive part. It begins with first retrieving the integer from the input parameter and then comparing that value with 0 by the command ifgt and then if it is, the code moves onto L1, if not, then the code moves to L2 and throw an arithmeticexception. It uses INVOKESPECIAL and ATHROW in order to do this.

If the value is greater than zero then the code moves into L1 which is where the actual recursion happens. The base case if first tested and if it fails then a command called INVOKESTATIC is called with the appropriate parameters. The parameters are loaded and calculated in the 4 lines just above the invokestatic command. After the recursion, a multiply command is called before return is executed.

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 that is generated from this tail recursive function is actually very similar and almost identical to the code in part “e”. Aside from the actual calculation returned, the exception as well as the recursion can be found in the code. Knowing that this is a tail recursive example, I expected to see code similar to a for loop. After doing some research online, I now realize that Java does not actually have tail recursive optimisation. If it did, Java would have never called the INVOKESTATIC command for a method and would have simply used a goto command until the condition was met. I even found a simple example online and tried that method, but it still used INVOKESTATIC.

**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, Harry)

P(Charles, William)

P(Diana, Harry)

P(Diana, William)

P(William, George)

P(William, Charlotte)

P(Kate, George)

P(Kate, Charlotte)

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

Male(Charles)

Male(Harry)

Male(William)

Male(Edward VII)

Male(George V)

Male(George VI)

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

**Father Rule**

Father(X,Y) := Parent(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!

**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.

After several attempts at creating my own mergeSort algorithm, I realized that I needed to find working code online for this question. Most of the code came from this website:

**http://www.java2novice.com/java-sorting-algorithms/merge-sort/**