**Shardul Virkar**

**CS 611: Principles of Programming Languages**

**Professor: Christelle Scharff**

**Homework 2 (10/16/2017)**

**sv38312n@pace.edu**

**Homework 2**

**Programming Languages Principles and Implementation**

**Instructions:**

* Due date: 10/16 (No late homework will be accepted.)
* 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, John W Backus, IBM, USA
* LISP: 1958, John Mc Carthy, MIT, USA
* COBOL: 1959, CODASYL (Conference/Committee on Data Systems Languages), USA
* ISETL: 1969, Jacob T Schwartz, New York University, USA
* Pascal: 1970, Niklaus Wirth
* Prolog: 1972, Alain Colmeraurer with Philippe Roussel, France
* C: 1972, Dennis Ritchie, Bell Labs, USA
* SML: 1973, Robin Milner, University of Edinburgh, Scotland
* ADA: 1980, Jean Ichbiah, USA
* C++: 1983, Bjarne Stroustrup, Bell Labs, USA
* EIFFEL: 1986, Bertrand Meyer
* PERL: 1987, Larry Wall, USA
* Python: 1991, Guido Van Rossum, CWI (Switching Mathematisch Centrum), Netherlands
* Java: 1995, James Gosling, Sun Microsystems, USA
* Ruby: 1998, Yukihiro “Matz” Matsumoto, Japan
* Kotlin: 2016, Jetbrains, 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

The method draw() has multiple forms i.e. signatures. The first one has no arguments whereas the second draw method has an argument. Both have the same method name but their signatures are different. This is known as Method Overloading, it is a compile time polymorphism.

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

Here, Circle is the super class and ColoredCircle is the subclass of Circle. The subclass extends from superclass, which means that it contains all the functionality of the superclass plus some functionality of its own. This is why we cannot write ‘d=c;’ as the sublass contains some things that are extra when compared to the superclass.

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?

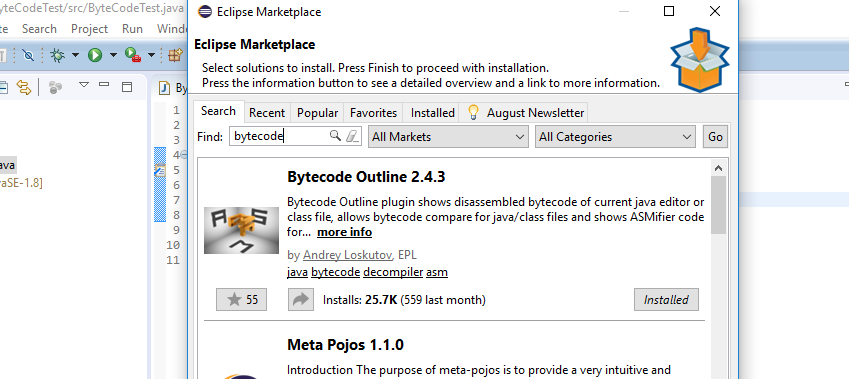
c = d;

c.draw();

Yes, we can write c = d; as the superclass is equal to subclass. When we execute the code below the draw() method of the subclass is assigned to superclass. When we call c.draw() method the draw method of d i.e ColoredCircle is called.

**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?

I am using version 4.7.1

1. What Java version are you using?

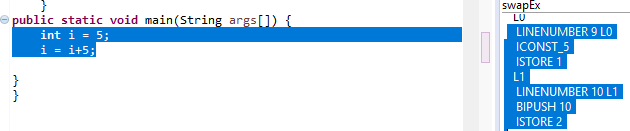
Java version "1.8.0\_144" which is java version 8, update 144.

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.



LINENUMBER 9 L0

ICONST\_5

ISTORE 1

L1

LINENUMBER 10 L1

BIPUSH 10

ISTORE 2

LINENUMBER: Indicates the current line number of the code

ICONST\_5: Push the Integer constant 0,1,2,3,4 or 5

ISTORE: Store integer in local variable

BIPUSH : Push one-byte signed integer

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;

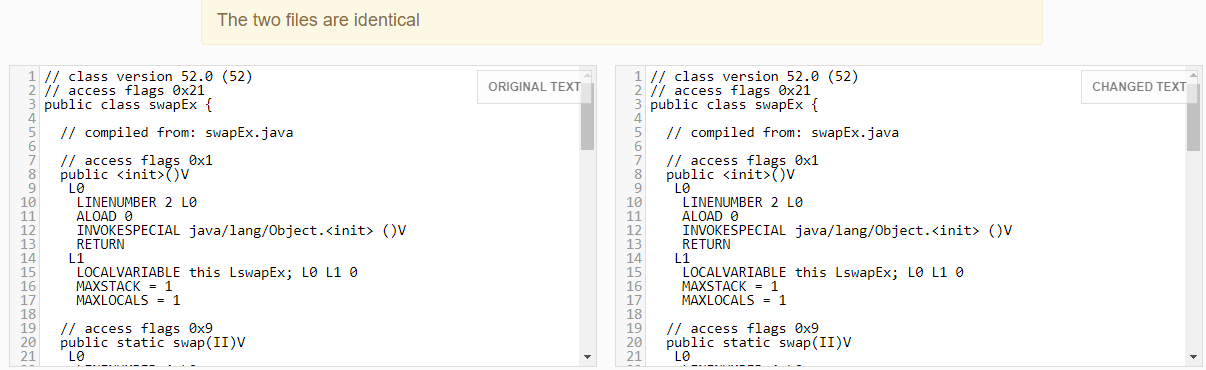
i++;

}

**return** sum;

}

As seen in the below screenshot the bytecode generated for both code is the same.



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

The bytecode generated by the function is:

// class version 52.0 (52)

// access flags 0x20

class Factorial {

// compiled from: swapEx.java

// access flags 0x8

static I facto

// access flags 0x0

<init>()V

L0

LINENUMBER 1 L0

ALOAD 0

INVOKESPECIAL java/lang/Object.<init> ()V

RETURN

L1

LOCALVARIABLE this LFactorial; L0 L1 0

MAXSTACK = 1

MAXLOCALS = 1

// access flags 0x9

public static fact(I)I

L0

LINENUMBER 4 L0

GOTO L1

L2

LINENUMBER 5 L2

FRAME SAME

GETSTATIC Factorial.facto : I

ILOAD 0

IMUL

PUTSTATIC Factorial.facto : I

L3

LINENUMBER 6 L3

IINC 0 -1

L1

LINENUMBER 4 L1

FRAME SAME

ILOAD 0

IFGT L2

L4

LINENUMBER 8 L4

GETSTATIC Factorial.facto : I

IRETURN

L5

LOCALVARIABLE n I L0 L5 0

MAXSTACK = 2

MAXLOCALS = 1

}

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).

**package** factorial;

**public** **class** fact {

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

// **TODO** Auto-generated method stub

**int** i,fact=1;

**int** number=5;//It is the number to calculate factorial

**for**(i=1;i<=number;i++){

fact=fact\*i;

}

System.***out***.println("Factorial of "+number+" is: "+fact);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*output\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Factorial of 5 is: 120

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*bytecode\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/ class version 52.0 (52)

// access flags 0x21

**public** **class** fact {

// compiled from: fact.java

// access flags 0x1

**public** <init>() : **void**

L0

LINENUMBER 3 L0

ALOAD 0: **this**

INVOKESPECIAL Object.<init> () : **void**

RETURN

L1

LOCALVARIABLE **this** fact L0 L1 0

MAXSTACK = 1

MAXLOCALS = 1

// access flags 0x9

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

L0

LINENUMBER 8 L0

ICONST\_1

ISTORE 2

L1

LINENUMBER 9 L1

ICONST\_5

ISTORE 3

L2

LINENUMBER 10 L2

ICONST\_1

ISTORE 1

L3

GOTO L4

L5

LINENUMBER 11 L5

FRAME APPEND [**int** **int** **int**]

ILOAD 2: fact

ILOAD 1: i

IMUL

ISTORE 2: fact

L6

LINENUMBER 10 L6

IINC 1: i 1

L4

FRAME SAME

ILOAD 1: i

ILOAD 3: number

IF\_ICMPLE L5

L7

LINENUMBER 13 L7

GETSTATIC System.out : PrintStream

NEW StringBuilder

DUP

LDC "Factorial of "

INVOKESPECIAL StringBuilder.<init> (String) : **void**

ILOAD 3: number

INVOKEVIRTUAL StringBuilder.append (**int**) : StringBuilder

LDC " is: "

INVOKEVIRTUAL StringBuilder.append (String) : StringBuilder

ILOAD 2: fact

INVOKEVIRTUAL StringBuilder.append (**int**) : StringBuilder

INVOKEVIRTUAL StringBuilder.toString () : String

INVOKEVIRTUAL PrintStream.println (String) : **void**

L8

LINENUMBER 16 L8

RETURN

L9

LOCALVARIABLE args String[] L0 L9 0

LOCALVARIABLE i **int** L3 L9 1

LOCALVARIABLE fact **int** L1 L9 2

LOCALVARIABLE number **int** L2 L9 3

MAXSTACK = 4

MAXLOCALS = 4

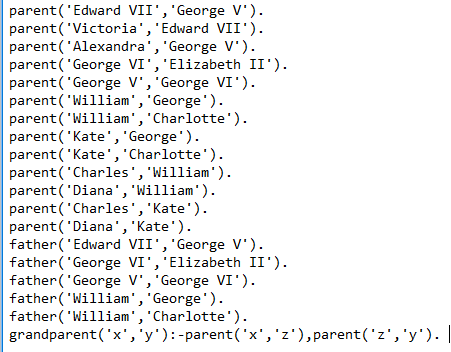
}

**References**

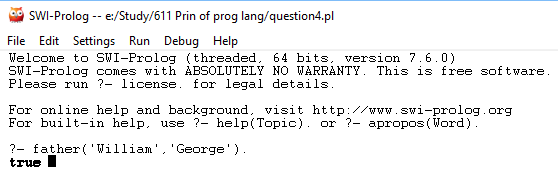
* The Java Virtual Machine Specification <https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf> (Java 8 SE)
* Java Bytecode Basics <http://www.javaworld.com/javaworld/jw-09-1996/jw-09-bytecodes.html> (1996)
* <http://www.beyondjava.net/blog/java-programmers-guide-java-byte-code/> (2015)

**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.



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

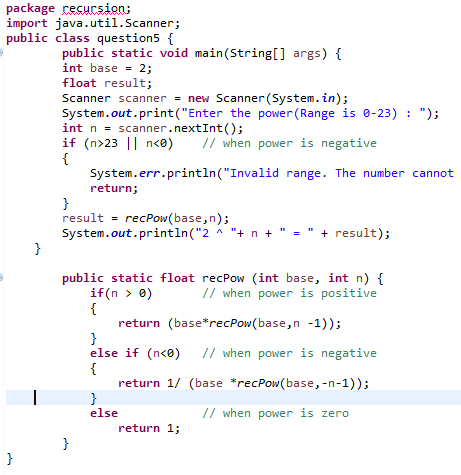


**Question 5:**

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.

class MergeExample

{

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

int[] merge(int arr[], int left, int middle, int right)

{

// Find sizes of two subarrays to be merged

int n1 = middle - left + 1;

int n2 = right - middle;

/\* Create temp arrays \*/

int L[] = new int [n1];

int R[] = new int [n2];

/\*Copy data to temp arrays\*/

for (int i=0; i<n1; ++i)

L[i] = arr[left + i];

for (int j=0; j<n2; ++j)

R[j] = arr[middle + 1+ j];

/\* Merge the temp arrays \*/

// Initial indexes of first and second subarrays

int i = 0, j = 0;

// Initial index of merged subarry array

int k = left;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy remaining elements of L[] if any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy remaining elements of R[] if any \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

return arr;

}

// Main function that sorts arr[l..r] using

// merge()

int[] split(int arr[], int l, int r)

{

if (l < r)

{

// Find the middle point

int m = (l+r)/2;

// Sort first and second halves

split(arr, l, m);

split(arr , m+1, r);

// Merge the sorted halves

return merge(arr, l, m, r);

}

return null;

}

/\* A utility function to print array of size n \*/

static void printArray(int arr[])

{

int n = arr.length;

for (int i=0; i<n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver method

public static void main(String args[])

{

int arr[] = {12, 11, 13, 5, 6, 7 , 5, 7, 8};

System.out.println("Given Array");

printArray(arr);

MergeExample ob = new MergeExample();

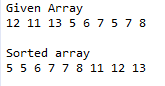
int[] mergedArray = ob.split(arr, 0, arr.length-1);

System.out.println("\nSorted array");

printArray(mergedArray);

}

}



REFERENCES: <https://www.obliquity.com/computer/fortran/history.html>

<http://groups.engin.umd.umich.edu/CIS/course.des/cis400/cobol/cobol.html>

<https://www.csee.umbc.edu/courses/graduate/631/Fall2002/COBOL.pdf>

<https://www.eiffel.com/resources/faqs/eiffel-language/#classic-how>

<https://www.britroyals.com/royaltree.asp>

<http://www.cs.toronto.edu/~sheila/384/w11/simple-prolog-examples.html>

<https://www.tutorialspoint.com/java/java_inheritance.htm>