Homework 2

John Mulcahy & Nick Krawczeniuk
CS361 Programming
Languages and Implementation
Professor Scharff
October 9th, 2017

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.
- 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 by John Backus and IBM in USA
- Lisp

    - 1958 by John McCarthy (designer) and Steve Russell and co. (developers) with Dartmouth college in USA
- Cobol
    - 1958 by Howard Bromberg and co at CODASYL for US Department of Defense in USA
- PASCAL
    - 1970 by Niklaus Wirth (solo but as member of International Federation of Information Processing) in Switzerland
- Prolog
    - 1972 by Alain Colmerauer at University of Aix-Marseille in France
- C
    - 1978 by Dennis Ritchie with AT&T Bell Labs in USA
- ADA
    - 1980 by Jean Ichbiah of CII Honeywell Bull (France) for US DoD (USA)
- C++
    - 1983 by Bjanre Stroustrup (solo) in Denmark
- ISETL
    - 1986 By Gary Levin at Clarkson University in USA
- EIFFEL
    - 1986 by Bertrand Meyer wiith Eiffel Software in France
- SML
    - 1987 by Robin Milner with University of Edinburgh LFCS in Scotland
- Perl
    - 1987 by Larry Wall with UNISYS in USA
- Python
    - 1991 by Guido van Rossum with Python Software Foundation in USA
- Java
    - 1995 by James Gosling with Sun Microsystems in USA
- Ruby
    - 1995 by Yukihiro Matsumoto and co (a fellow of Rakuten Institute of Technology) in Japan
- Kotlin
    - 2011 by JetBrains (lead dev Dmitry Jemerov) in 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
	}
}
```

a) Explain polymorphism on the code above.

Polymorphism is the term used to explain subclasses defining their own individual behaviors unique from the parent class. In the example, ColoredCircle is a subclass of Circle, defining specific characteristics of said circle, opposed to when Circle is called, and a hollow circle is drawn.

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

We can say d=c, because ColoredCircle is a subclass of Circle. It contains all of the traits of circle, plus an additional (color). D is a child class to C, therefore it can be equal.

c) 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?

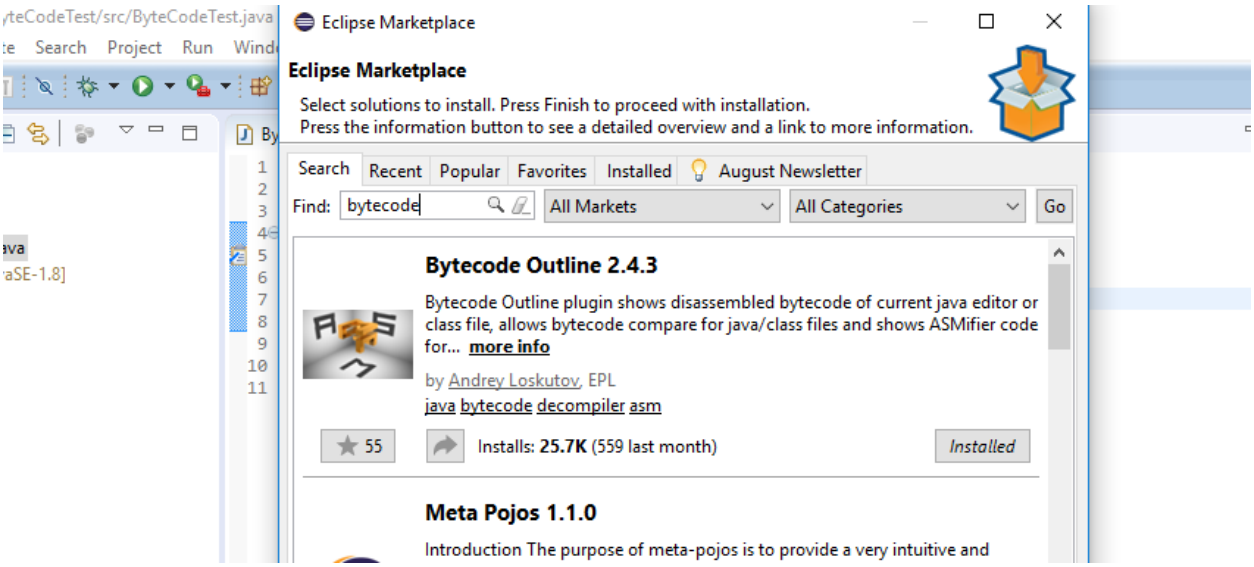
```
c = d;
c.draw();
```

We cannot say c=d, because circle does not contain all of the additional traits of a ColoredCircle; just the traits defined in Circle. If executed, it will simply draw a regular circle, not a colored circle. This is because "c" does not have the allocated memory/variables for the colored trait. A parent class cannot be equal to a child class, as it is not necessarily the same. The draw that is being called is type Circle's. When it is run there will be a compiler or runtime error, depending on the location of the check.

*referenced article for specific information on error: https://stackoverflow.com/questions/43296084/why-cant-i-assign-a-parent-class-to-a-variable-of-subclass-type

Question 3:

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

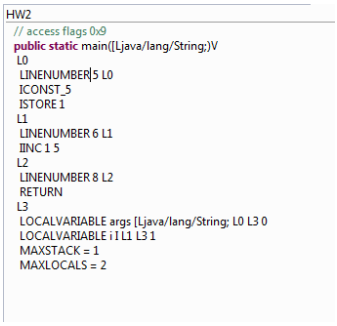


[Dr. Scharff tested with the Neon version of Eclipse and with Eclipse Marketplace Byte Outline 2.4.3 plugin and it works!]

a) What Eclipse version are you using?
Mars.2 Release (4.5.2)

b) What Java version are you using?
java.runtime.version=1.8.0_73-b02

c) 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.



Essentially, a variable is initiated, and the value 5 is stored in it. Then, it is incrementally increased by five in the next line.

```
// access flags 0x9
public static main([Ljava/lang/String;)V
 L0
  LINENUMBER 5 L0
  ICONST_5 //Load value 5 into stack
  ISTORE 1 //store a long value in a local variable 1
 L1
  LINENUMBER 6 L1
```

```
 IINC 1 5 //increment local variable 1 by five
L2
 LINENUMBER 8 L2
 RETURN
L3
 LOCALVARIABLE args [Ljava/lang/String; L0 L3 0
 LOCALVARIABLE i I L1 L3 1
 MAXSTACK = 1 //max operand stack depth (1)
 MAXLOCALS = 2 //number of local variables
```

d) 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;
}
```

```
HW2
// class version 52.0 (52)
// access flags 0x21
public class HW2 {

  // compiled from: HW2.java

  // access flags 0x1
  public <init> ()V
   L0
    LINENUMBER 2 L0
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN
   L1
    LOCALVARIABLE this LHW2; L0 L1 0
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 0x9
  public static sum_for(I)I
   L0
    LINENUMBER 4 L0
    ICONST_0
    ISTORE 1
   L1
    ICONST_0
    ISTORE 2
   L2
    LINENUMBER 5 L2
    ICONST_0
    ISTORE 1
```

```
HW2
    GOTO L3
   L4
    LINENUMBER 6 L4
   FRAME APPEND [I I]
    ILOAD 2
    ILOAD 1
    IADD
    ISTORE 2
   L5
    LINENUMBER 5 L5
    IINC 1 1
   L3
   FRAME SAME
    ILOAD 1
    ILOAD 0
    IF_ICMPLE L4
   L6
    LINENUMBER 8 L6
    ILOAD 2
    IRETURN
   L7
    LOCALVARIABLE n I L0 L7 0
    LOCALVARIABLE i I L1 L7 1
    LOCALVARIABLE sum I L2 L7 2
    MAXSTACK = 2
    MAXLOCALS = 3

  // access flags 0x9
  public static sum_while(I)I
   L0
    LINENUMBER 12 L0
```

```
HW2
    ICONST_0
    ISTORE 1
   L1
    ICONST_0
    ISTORE 2
   L2
    LINENUMBER 13 L2
    GOTO L3
   L4
    LINENUMBER 14 L4
   FRAME APPEND [I I]
    ILOAD 2
    ILOAD 1
    IADD
    ISTORE 2
   L5
    LINENUMBER 15 L5
    IINC 1 1
   L3
    LINENUMBER 13 L3
   FRAME SAME
    ILOAD 1
    ILOAD 0
    IF_ICMPLE L4
   L6
    LINENUMBER 17 L6
    ILOAD 2
    IRETURN
   L7
    LOCALVARIABLE n I L0 L7 0
    LOCALVARIABLE i I L1 L7 1
    LOCALVARIABLE sum I L2 L7 2
    MAXSTACK = 2
    MAXLOCALS = 3
}
```

To the typical user, these functions do the same thing- take a number (n) and add up all numbers from 0 to n (0+1+2+3+4+5….+n). The only difference is the type of loop used from this arithmetic-

first, a for loop, and next, a while loop. We can gain a lot more insight into the process of these functions by examining the bytecode. By examining the bytecode, we see that both cases are handled exactly the same. The key parts of the code are identical to each other. Functionally, both loops are handled by the computer the same way.

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

```
public static int factorial(int num)
{
	if (num == 0 || num == 1)
		return 1;
	else
		return num * factorial(num-1);
	}
```

```
// access flags 0x9
public static factorial(I)I
 L0
  LINENUMBER 21 L0
  ILOAD 0
  IFEQ L1
  ILOAD 0
  ICONST_1
  IF_ICMPNE L2
 L1
  LINENUMBER 22 L1
 FRAME SAME
  ICONST_1
  IRETURN
 L2
  LINENUMBER 24 L2
 FRAME SAME
  ILOAD 0
  ILOAD 0
  ICONST_1
  ISUB
  INVOKESTATIC HW1.factorial (I)I
  IMUL
  IRETURN
 L3
  LOCALVARIABLE num I L0 L3 0
  MAXSTACK = 3
  MAXLOCALS = 1
```

```
// access flags 0x9
public static factorial(I)I
 L0
  LINENUMBER 21 L0
  ILOAD 0 //load value into local variable 0
  IFEQ L1 //if value 0 branch to instructions at L1
  ILOAD 0 //load value
  ICONST_1 //load value 1 into stack
  IF_ICMPNE L2 //if ints not equal, go to L2
 L1
  LINENUMBER 22 L1
 FRAME SAME
  ICONST_1 //load int value 1 into stack
  IRETURN //return int from L1
 L2
  LINENUMBER 24 L2
 FRAME SAME
  ILOAD 0 //load 0
```

```
ILOAD 0 //load 0
ICONST_1 //load value 1 into stack
ISUB //int substract (val1-val2=result)
INVOKESTATIC HW1.factorial (I)I //invoke static method, put result on stack
IMUL //multiply two ints
IRETURN //return value
L3
LOCALVARIABLE num I L0 L3 0
MAXSTACK = 3
MAXLOCALS = 1
```

By breaking down this code, we see recursion broken down into its basic form. The problem is broken down into two paths – L1 and L2. If the variable is equal to 1, go to L1 (where final answer is returned). If greater than 1, the number is multiplied to the sum, reduced, and the function is called again. This is obviously very similar to the original Java, just broken down into smaller steps.

a) 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).

```
// class version 51.0 (51)
// access flags 0x21
public class ByteCodeTest {

  // compiled from: ByteCodeTest.java

  // access flags 0x1
  public <init>()V
   L0
    LINENUMBER 2 L0
    ALOAD 0 //load a reference onto the stack from a local variable
    INVOKESPECIAL java/lang/Object.<init> ()V //Invoke instance method on object and puts the result in stack
    RETURN
   L1
    LOCALVARIABLE this LByteCodeTest; L0 L1 0
    MAXSTACK = 1
    MAXLOCALS = 1
```

```
// access flags 0x0
fact(II)I
 L0
  LINENUMBER 5 L0
  ILOAD 1 //load value into local variable 1
  IFNE L1 //if not 0, branch to branchoffset
 L2
  LINENUMBER 6 L2
  ILOAD 2 //load into local variable 2
  IRETURN //return int
 L1
  LINENUMBER 8 L1
 FRAME SAME
  ALOAD 0 //load reference onto stack from local var 0
  ILOAD 1 //load variable from local variable 1
  ICONST_1 //load constant value 1 onto stack
  ISUB //integer subtraction
  ILOAD 2 //load from variable 2
  ILOAD 1 //load value from variable 1
  IMUL //multiply integers
  INVOKEVIRTUAL ByteCodeTest.fact (II)I //invoke virtual method on object and put result on stack
  IRETURN //return integer
 L3
  LOCALVARIABLE this LByteCodeTest; L0 L3 0
  LOCALVARIABLE i I L0 L3 1
  LOCALVARIABLE acc I L0 L3 2
  MAXSTACK = 4
  MAXLOCALS = 3
        }
```

Used for reference: http://www.drdobbs.com/jvm/tail-call-optimization-and-java/240167044

Essentially, the difference between this and the previous recursive function is that there is no base case for the recursion to start from. The function calls on itself until the very end. This optimizes the code to the fullest, eliminating all unnecessary code that would typically be repeated in the execution in the code, saving them for the very end (the only time they are necessary). Recursive code is replaced with a loop, as stated in the link above, “reducing pressure from the stack”.

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:

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

```
:- 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).

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

Question 5:

Write a **recursive** function *recPow* that computes 2^n for n >= 0 in Java. The function will have the following profile:

```
public static int recPow(int n)
```

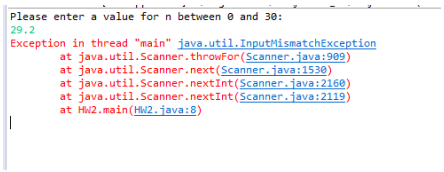
The function must consider all cases and be tested exhaustively. Show your testing!

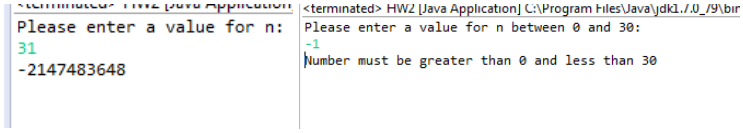
```
import java.lang.Math;
import java.util.Scanner;
public class HW2 {

	public static void main(String[] args){
		Scanner user_input = new Scanner(System.in);
		System.out.println("Please enter a value for n between 0 and 30:");
		int n = user_input.nextInt(); //Throws error if number is not an int
(tested)
		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"); //prints
message if problem is not computable
		return 0; //0 is returned but is not printed to user
	}
	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).

```
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);//recursion used by the sort
            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){
      //split of array into smaller arrays
      int leftsize = middle - splitlow +1;
      int rightsize = splithigh - middle; //get sizes of each array


      int[] left  = new int[leftsize + 1];
      int[] right = new int[rightsize + 1]; //split array in two

      left[leftsize] = Integer.MAX_VALUE;
      right[rightsize] = Integer.MAX_VALUE;//simplified method given in cited
tutorial to get the largest of each array


      for(int i = 0; i < leftsize; i++)
            left[i] = array[splitlow +i];//left sort
      for(int i = 0; i < rightsize; i++)
            right[i] = array[i+middle+1];//right sort



      //now the two arrays will be combined
      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++;
            }
      }
}
}
```

The function must be tested exhaustively. Show your testing! *Testing for this left in main. Here is sorted arrays of various integers and sizes

```
<terminated> mergeSort [Java Applicat
-23 -7 1 6 9 10 12 32 56
-23 1 2 3 4 9 10 78 87 98 102
-89 6 8 9 10 12 12 15 56 87
-45 0 1 4 4 9 10 12 25
-89 9 9 55
54 66
```

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

The following links were used to assist in the development of my code:

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