Saurabhi Shewale

ss19454n@pace.edu

Manank Shastri

ms98408n@pace.edu

Abstract

This homework consists of History of programming languages, Byte code generation, PROLOG and Recursive functions

Homework 2

Programming Languages Principles and Implementation

**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.

**ANS:**

* Fortran - 1957, John Backus, IBM, USA
* Lisp - 1958, John McCarthy, MIT, USA
* Cobol - 1959, CODASYL(Conference/Committee on Data Systems Languages), USA
* ISETL - 1986, Gary Levin, Clarkson University, USA
* PASCAL - 1970, Niklaus Wirth, named after Blaise Pascal — a French mathematician
* Prolog - 1972, (Alain Colmerauer with Philippe Roussel), France
* C - 1972, Dennis Ritchie, Bell labs, USA
* SML - Robin Milner & others at the University of Edinburgh, First appeared 1973; 44 years ago
* ADA – 1980, Jean Ichbiah
* C++ - 1983, Bjarne Stroustrup, Bell labs, USA
* EIFFEL - 1986, Bertrand Meyer.
* Perl - 1987 , Larry Wall, USA
* Python - 991 ,Guido Van Rossum, CWI(Stichting 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.

**ANS:**

Class containing method with same name and different signature is called Polymorphism.

Different signatures such as number of parameters, data type of parameters and sequence of parameters.

In the code above method (1) and (2) are of same class with same name but one has parameter passing in it.

Method (3) is an over-ridden method.

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

**ANS:**

Yes. We can write c=d. Because of Upcasting and Downcasting concept in Java. By casting you are not actually changing the object itself, you are just labeling it differently.

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?

**ANS:**

Yes. We can write c=d. Because of Upcasting and Downcasting concept in Java. By casting you are not actually changing the object itself, you are just labeling it differently.

After we call these codes:

c = d; - it assigns the d object to reference of c;

c.draw(); - after assigning c=d, c.draw() will actually execute d.draw(), (get method gets executed always).

**Question 3:**

1. What Eclipse version are you using?

Oxygen Release (4.7.0)

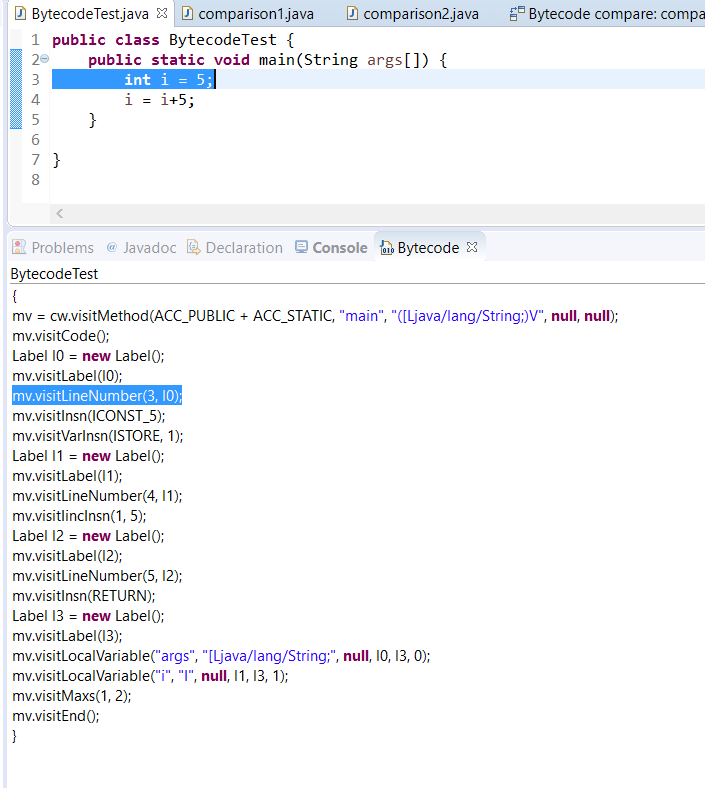
1. What Java version are you using?

Java SE 8

1. What is the Bytecode generated by the following statements?

int i = 5;

i = i+5;



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;

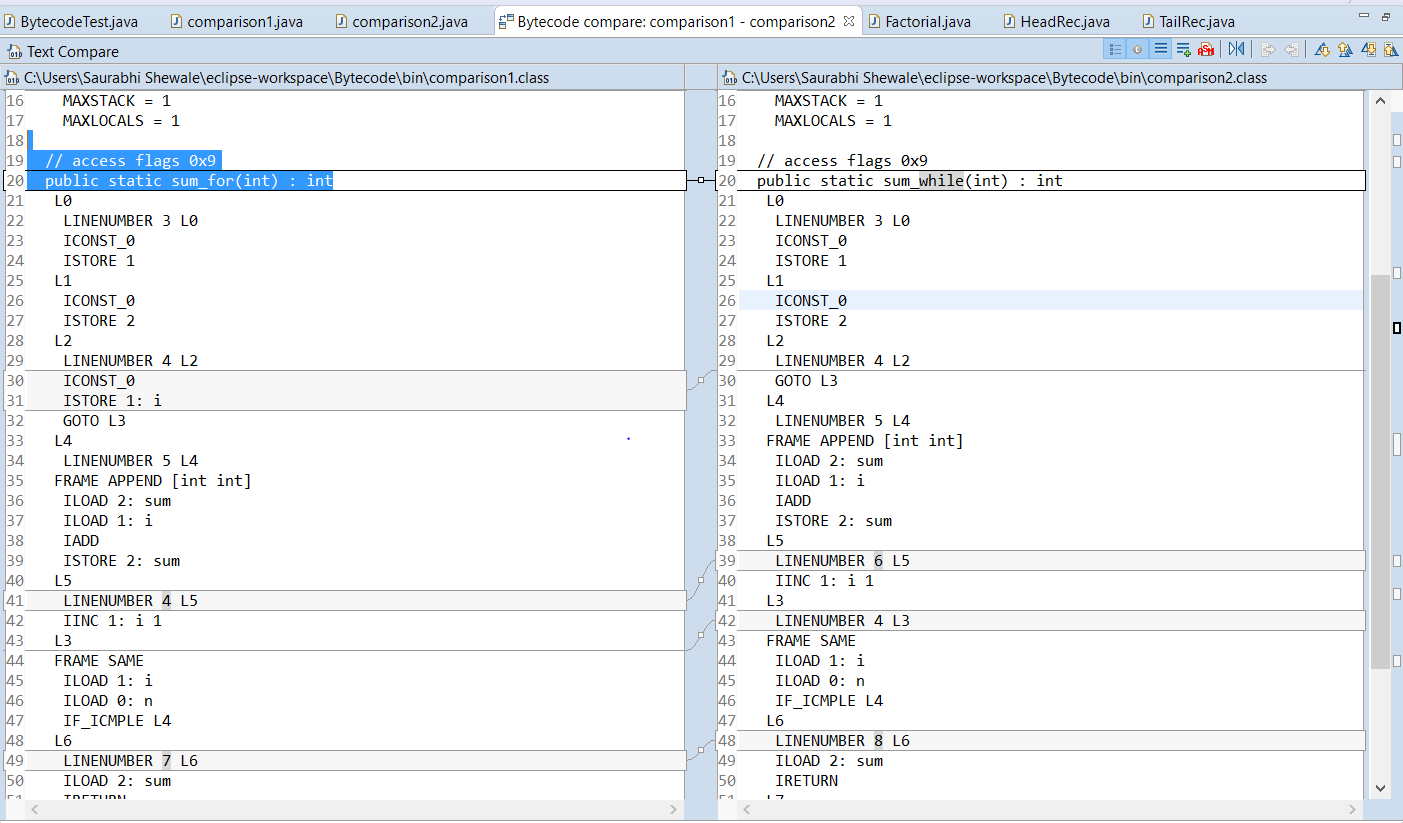
i++;

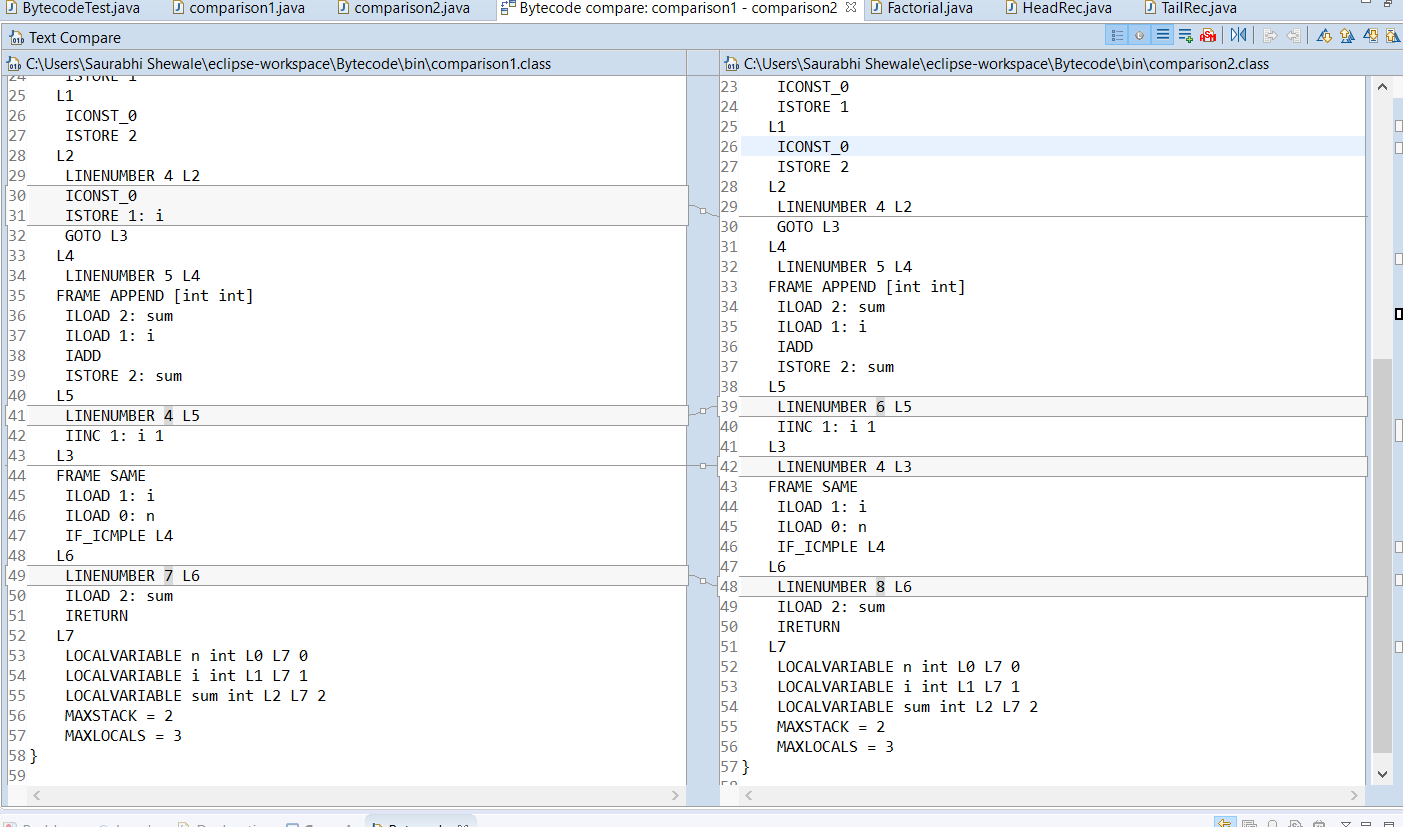
}

**return** sum;

}

**ANS:**





There is no difference in byte code of above two functions, but the performance varies if we use different compliers to generate byte code.

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

**ANS:**

**public** **class** Factorial {

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

}

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

**if** (n == 0)

**return** 1;

**else**

**return** n \* *fact*(n-1);

}

}







1. Choose a tail recursive function and describe the bytecode generated by this function. Compare the code that is generated with the code of a recursive function. What do you observe?

**ANS:**

// Program for recursive function

**public** **class** TailRec {

**public** **void** tail(**int** n){

**if**(n == 1)

**return**;

**else**

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

tail(n-1);

}

}

//Program for Tail Recursion

**public** **class** HeadRec {

**public** **void** head(**int** n){

**if**(n == 0)

**return**;

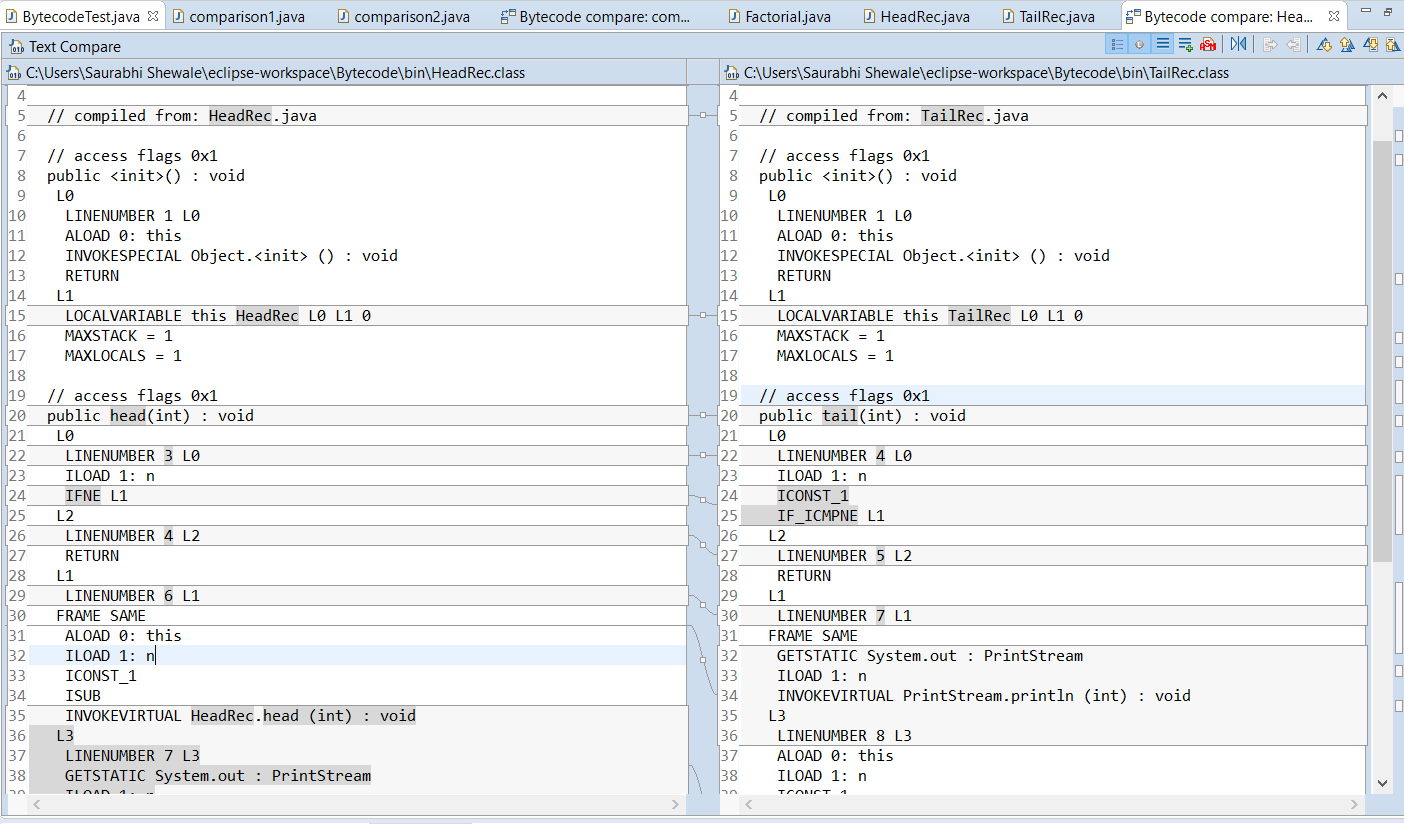
**else**

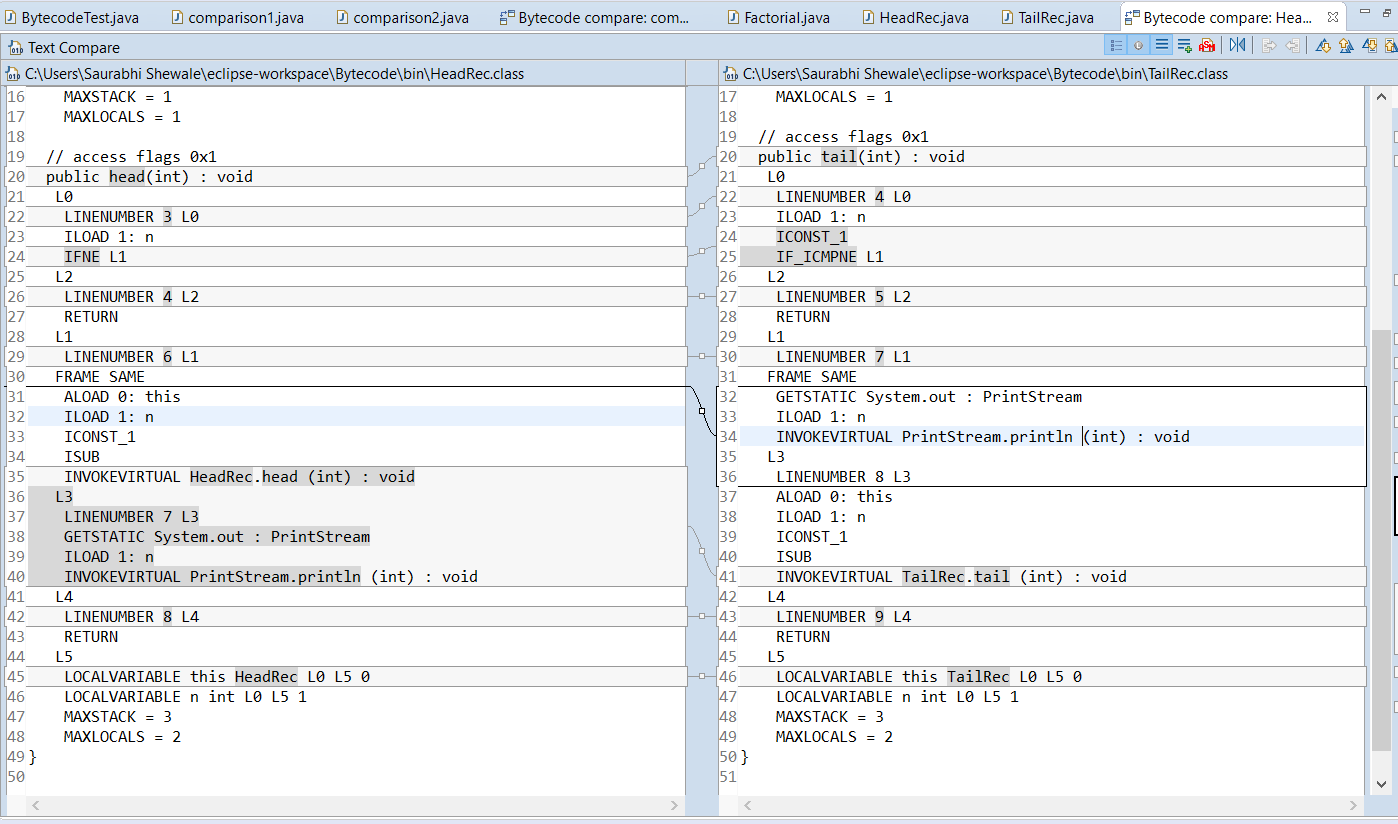
head(n-1);

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

}

}





**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.

**ANS:**

parent('Edward VII','George V').

parent('Victoria','Edward VII').

parent('Alexandra','George V').

parent('George VI','Elizabeth II').

parent('George V','George VI').

parent('William','George').

parent('William','Charlotte').

parent('Kate','George').

parent('Kate','Charlotte').

grandparent('x','y'):-parent('x','z'),parent('z','y').

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

**ANS:**

father('Edward VII','George V').

father ('George VI','Elizabeth II').

father ('George V','George VI').

father ('William','George').

father ('William','Charlotte').

**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!

**ANS:**

**import** java.util.Scanner;

**public** **class** RecursivePowerOf2 {

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

**int** base = 2;

**double** result;

Scanner sc = **new** Scanner(System.***in***);

System.***out***.print("Enter the power of 2: ");

**int** n = sc.nextInt();

result = *recPow*(base,n);

**if** (result != 0)

System.***out***.println("2 ^ "+ n + " = " + result);

}

**public** **static** **double** recPow (**int** base, **int** n) {

**if** (n > 23) {

System.***out***.println("Value of Power greater than 23");

**return** 0;

}

**else**

**if**(n > 0 && n < 24)

**return** (base\**recPow*(base,n -1));

**else**

**if** (n<0)

**return** 1/ (base \**recPow*(base,-n-1));

**else**

**return** 1;

}

}

**Question 6:**

Write a **recursive** function that implements merge sort in Java. . The function will have the following profile:

public static int[] mergeSort(int[])

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.

**ANS:**

**public** **class** RecursiveMergeSort {

**public** **static** **void** merge(**int** []a, **int** []b, **int** []c , **int** m, **int** n, **int** k){

**if** (n<0){

**if** (m<0) {

**return**;

}

c[k--] = a[m--];

}

**else**

**if** (m<0){

**if** (n<0){

**return**;

}

c[k--] = b[n--];

}

**else**{

c[k--] = (a[m]<b[n]) ? b[n--] : a[m--];

}

*merge*(a,b,c,m,n,k);

}

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

**int** []a={14,2,16,27,81,9};

**int** []b={20,0,11,46,18,0};

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

*merge*(a, b, c, a.length-1, b.length-1, c.length-1);

**for** (**int** i = 0; i < c.length; i++) {

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

}

}

}

References:

<https://stackoverflow.com/questions/21426688/the-difference-between-head-tail-recursion>