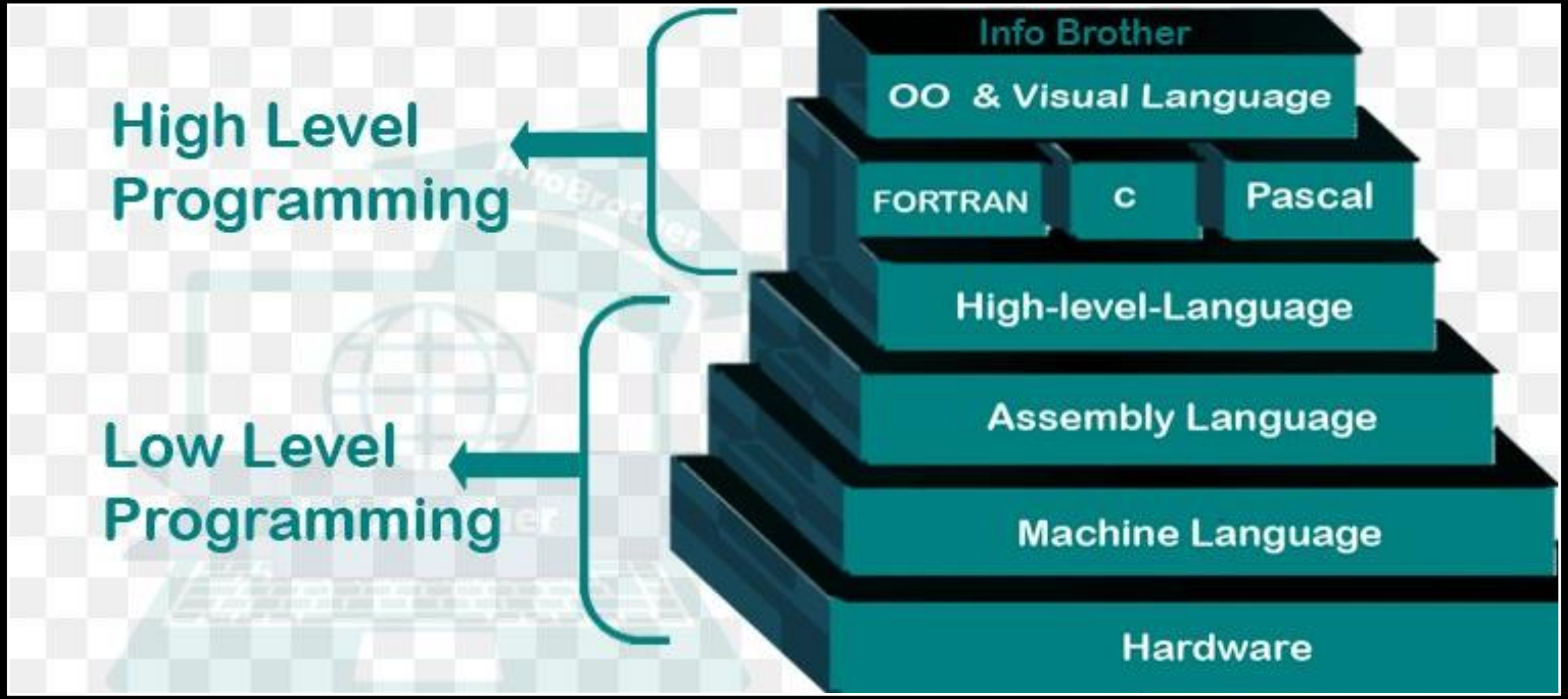
Java Midterm

Siwat Sirichai



Level of Programming Languages

Java Pre-compile Code vs ISAs Code

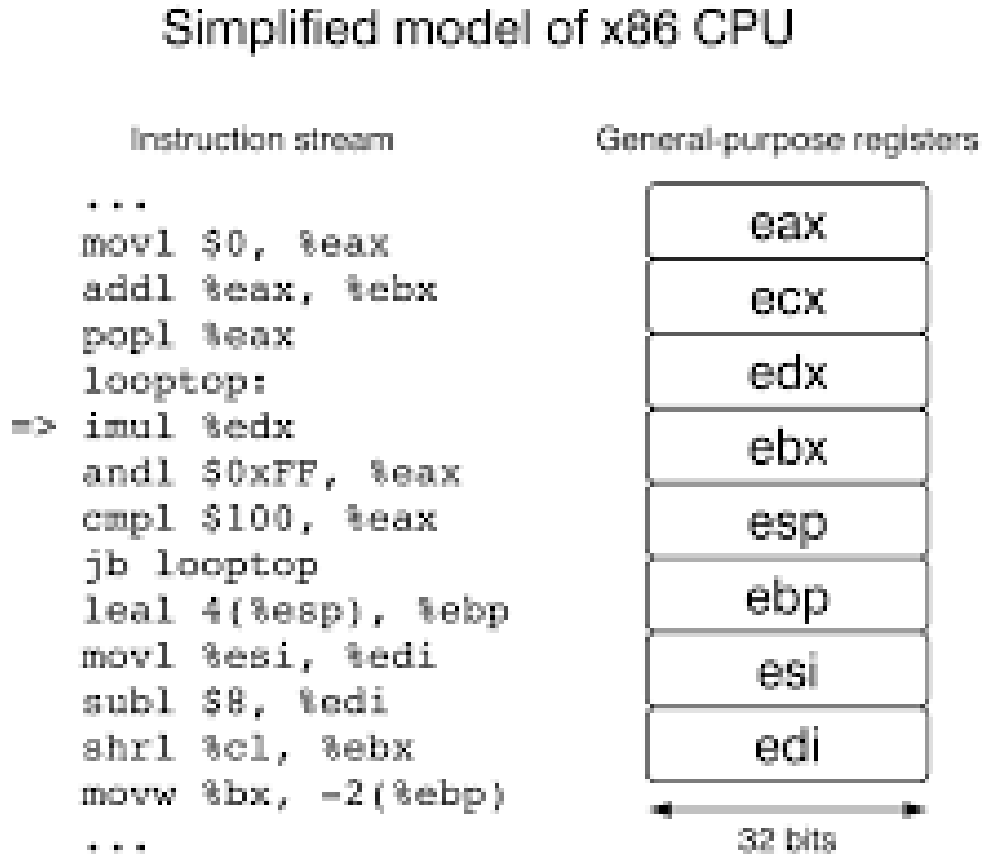
JavaExample.java

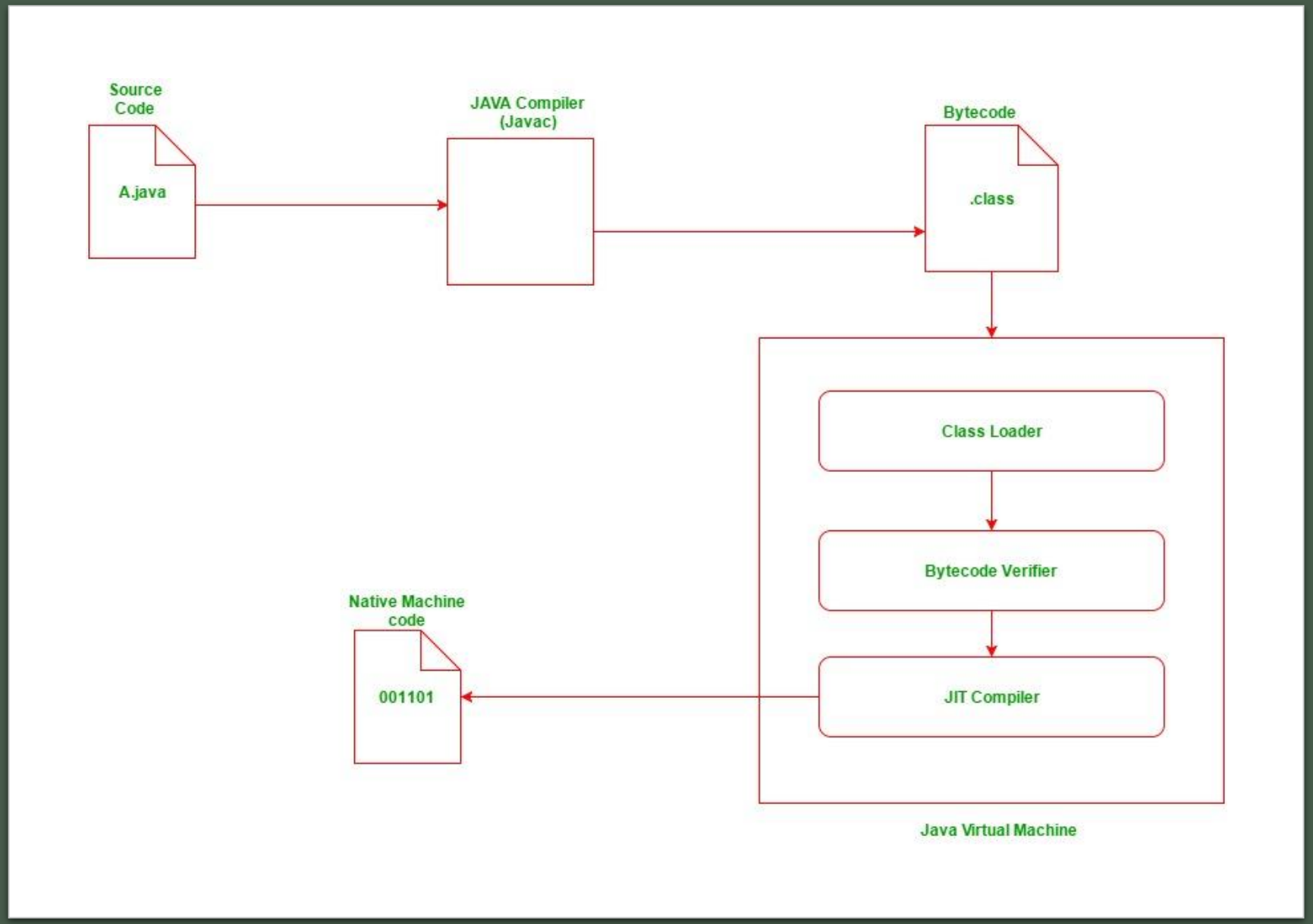
```java
package com.beginnersbook;
public class JavaExample {

    public static void main(String[] args) {
        String str = "BeginnersBook";
        int vcount = 0, ccount = 0;

        //converting all the chars to lowercase
        str = str.toLowerCase();
        for(int i = 0; i < str.length(); i++)
        {
            char ch = str.charAt(i);
            if(ch == 'a' || ch == 'e' || ch == 'i'
                || ch == 'o' || ch == 'u') {
                vcount++;
            }
            else if((ch >= 'a'&& ch <= 'z')) {
                ccount++;
            }
        }
        System.out.println("Number of Vowels: " + vcount);
        System.out.println("Number of Consonants: " + ccount);
    }
}
```

Problems @ Javadoc Declaration Console Progress Cover

<terminated> JavaExample [Java Application] /Library/Java/JavaVirtualMachines/jdk-9.0

```
Number of Vowels: 5
Number of Consonants: 8
```





How a Java program get executed

JDK Usage

- javac $FILE_NAME.java Compile .java File into Java Bytecode (.class)
- java $CLASS_NAME $PARAMETER1 $PARAMETER2 … Run the Main method of the Compile Bytecode (.class) named $CLASS_NAME in the current directory and pass in parameters as a String array.

Bonus: Bash / MSDOS Basic commands

- cd $DIRECTORY navigate to the $DIRECTORY directory
- cd ./$DIRECTORY navigate the $DIRECTORY directory inside the current directory
- cd ../ go back one directory.

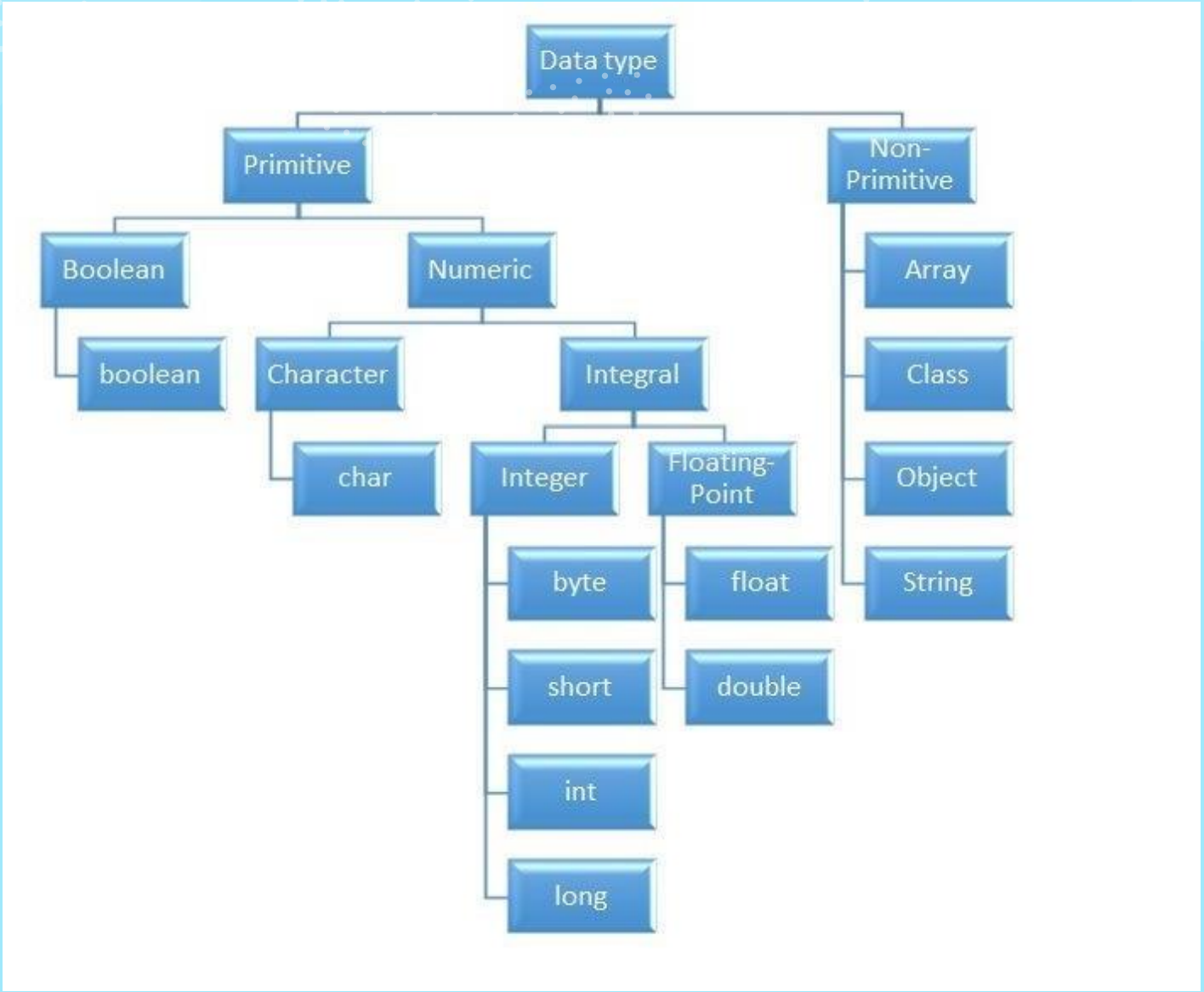
```
class MyJavaClass {
        public static void main(String[] args){
                //Your Code Here
        }
}
```

Basic Entry Class Structure

The basics statements

- System.out.println(String message); Print out a message then start a new line
- System.out.print(String message); Print out a message

Data type in Java



Datatype Concepts

Non-primitive is a pointer pointing to an object in memory.

Primitive stores the data directly in memory.

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\','\n',' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

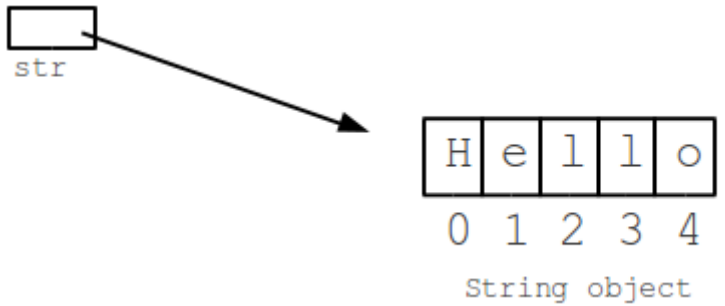
Datatype Conversion

- Numerical Assignment Hierarchy

byte → short → int → long → float → double

- String to char (text is of type string) text.charAt(int index);
- char to String Character.toString(char character);
- Primitive Type to int [Type].parse[Type](String text);

Ex. Integer.parseInt(String numberic_text); will return an Integer.



Datatype of Expressions

String + anything = String

int + double = double

int / double = double

int / int = int WARNING: RESULT WILL ALWAYS BE ROUNDED DOWN!

int / 0 = throw ArithmeticException!

double / 0 = Infinity

0.0 / 0.0 = NaN

Boolean datatypes

- 1==1 | true (one is equal to one)
- 1!=1 | false (one is not not equal to one)
- true && true | true (true logical and true)
- true && false | false (true logical and false)
- true || true | true (true logical or true)
- true || false | false (true logical or false)

Operators

Precedence	Operator	Operand type	Description
1	++,	Arithmetic	Increment and decrement
1	+, -	Arithmetic	Unary plus and minus
1	~	Integral	Bitwise complement
1	!	Boolean	Logical complement
1	(*type*)	Any	Cast
2	*, /, %	Arithmetic	Multiplication, division, remainder
3	+, -	Arithmetic	Addition and subtraction
3	+	String	String concatenation
4	<<	Integral	Left shift
4	>>	Integral	Right shift with sign extension
4	>>>	Integral	Right shift with no extension
5	<, <=, >, >=	Arithmetic	Numeric comparison
5	instanceof	Object	Type comparison
6	==, !=	Primitive	Equality and inequality of value
6	==, !=	Object	Equality and inequality of reference
7	&	Integral	Bitwise AND
7	&	Boolean	Boolean AND
8	^	Integral	Bitwise XOR
8	^	Boolean	Boolean XOR
9	|	Integral	Bitwise OR
9	|	Boolean	Boolean OR
10	&&	Boolean	Conditional AND
11	||	Boolean	Conditional OR
12	?:	N/A	Conditional ternary operator
13	=	Any	Assignment

CATCH: The confusing ++ and -- position

int x = 3;

- x+++1 = 4 x+1 is evaluated before the statement x++
- ++x+1 = 5 ++x is executed before the statement x+1 is evaluated

CATCH: Comparing non-primitive type with the == or != operator

String a = “Hello”;

a+=“ World”;

a==“Hello World” | this is false (== Compare the object’s address)

a.equals(“Hello World”) | this is true (.equals compare the value)

DON’T USE == or != WITH A PRIMITIVE TYPE, RESULTS MAY VARIES

Statement Shortcuts

x += 3 is equal to x = x+3

x -= 3 is equal to x = x-3

x *= 3 is equal to x = x*3

x /= 3 is equal to x = x/3

CATCH: Fucking Floating Points

double x = 0.1;

x += 0.1;

x += 0.1;

x==0.3 //THIS IS FALSE FOR SOME FUCKING REASONS ……

So when comparing floating point we usually use

Math.abs(x-0.3) < Math.power(10,-9)

Instead of

x == 0.3

Scanner Object Creation from Scanner Class

Scanner name = new Scanner(source);

Source list:

System.in | Terminal / CMD

new File(String path) | from a file

Getting data from source using Scanner

```
Scanner sc = new Scanner(Source);
sc.nextInt(); //get an int until the delimiter is reached
sc.nextDouble(); //get a double until the delimiter is reached
sc.next(); //get a String until the delimiter is reached
sc.nextLine(); //get a String until \n is reached
```

CATCH: nextLine() after nextInt() or nextDouble()

Source: “1\n2\nHello”

Scanner sc = new Scanner(Source);

sc.nextInt(); // return 1

sc.nextInt(); // return 2

sc.nextLine(); //Return “” (Nothing)

sc.nextLine(); // Return “Hello”

TIPS: Changing the delimiter

//The default delimiter is “ “ (a space)

Source: “abc,def,ghj”

Scanner sc = new Scanner (Source).useDelimiter(‘,’);

sc.next(); | return abc

sc.next(); | return def

sc.next(); | return ghj

String Method #1

String a = “ABCDEFGHI”

a.equals(“ABCDEFGHI”); | return true

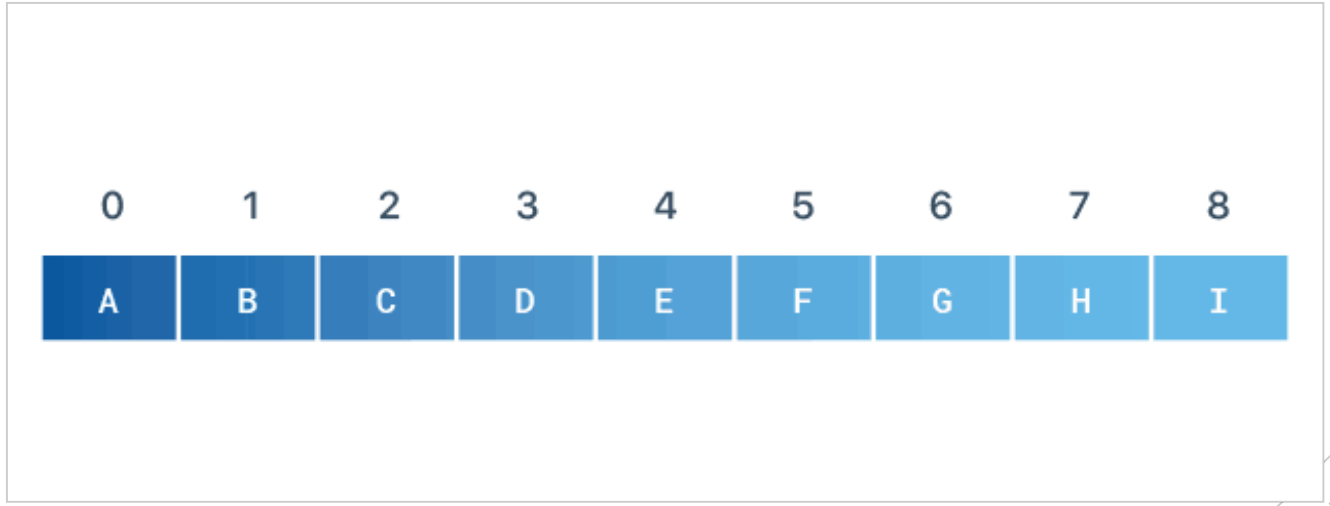
//like == for primitive type but for String

a.length(); | return 9

//return last_index+1

a.charAt(int index); | return char at index

if index = 4 | return E



String Method #2

String a = “ABCDEFGHI”

a.substring(int beginning_index); //beginning_index is inclusive

//return a new string with content from index [beginning_index,String.length())

if beginning_index = 5 | return FGHI

a.substring(int beginning_index,int ending_index);

// return a new string with content from index [beginning_index,ending_index)

//beginning_index is inclusive but ending_index is exclusive;

if beginning_index = 3, ending_index = 7 | return DEFG

Try-Catch Exception Handling

```
try {
        System.out.println(0/0); //This Statement generate an AtithmeticExcpetion
} catch (ArithmeticException e){
        System.out.println(“ERROR!”);
} catch (Exception e){ //one try block can have multiple catch block
        e.printStackTrace();
}
```

RESULT: ERROR!

General Rules for Exceptions

- Exception Thrown from main will results in a crash
- methods that call methods that uses the throws Exception keyword must be able to handle that exception. throws Exception declare that that method can throws Exception and must be handled

```
public static void main(String[] args) {
        System.out.println(dividebyzero());
}
public static int dividebyzero(){
        return 0/0;
}
```

RESULT: ARITHMETIC EXCEPTION

Throwing Exception #1

Throwing Exception #2

```
public static void main(String[] args) {
        System.out.println(dividebyzero());
}
public static int dividebyzero() throws ArithmeticExcpetion{
        return 0/0;
}
```

RESULT: Compile Error

```
public static void main(String[] args) {
        try {
                System.out.println(dividebyzero());
        } catch(ArithmeticException e){
                System.out.println("ERROR!");
        }
}
public static int dividebyzero() throws ArithmeticExcpetion{
        return 0/0;
}
```

RESULT: "ERROR!"

Throwing Exception #3

Throwing Exception #4

```
public static void main(String[] args) {
        try {
                System.out.println(dividebyzero());
        } catch(ArithmeticException e){
                System.out.println("ERROR!");
        }
}
public static int dividebyzero(){
        return 0/0;
}
```

RESULT: "ERROR!"

Decision Making with if

Syntax:

```
if(boolean expression) {
        statement1;
        statement2;
} else {
        statement3;
}
```

General rules for if-else

- if with one statement does not need {}

```
if (i==1)i++;
```

is equal to

```
if (i==1) {
        i++;
}
```

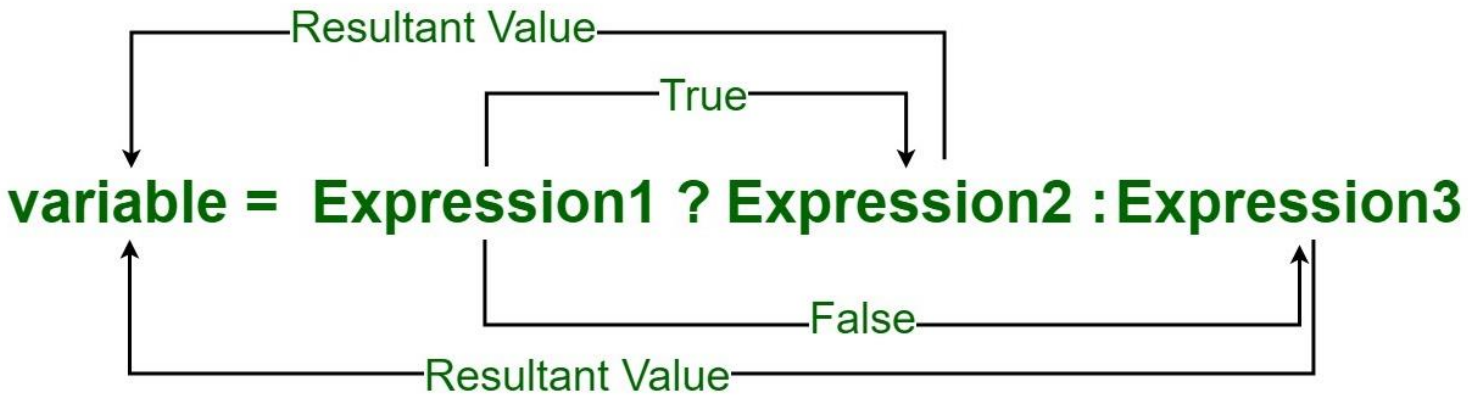
- if can have else but is not necessary
- if-else chain can be written as else if

```
if (i==1) i++;
else {
        if(i==2)i--;
}
```

is equal to

```
if(i==1)i++;
else if(i==2)i--;
```

Conditional or Ternary Operator (?:) in C/C++



Decision Making with Ternary Operator

Decision Making with Ternary Operator

```
int x=1;
if (x==1)x=0;
else x=4;
```

is equal to

```
int x=1;
x=(x==1)?0:4;
```

Iteration – While Loop

```
while (expression) {
        statement1;
        statement2;
        …
}
```

- Expression will be evaluated first before entering the while loop.
- Statement will be run until the expression is false

Iteration – While Loop Example

```
int x = 0;
while (x<=10) {
        System.out.print(x+” ”);
        x++;
}
```

RESULT: 0 1 2 3 4 5 6 7 8 9 10

Iteration – While Loop

```
do {
        statement1;
        statement2;
        …
} while(expression);
```

- Statement(s) run once before evaluating the expression
- Statement will continue to run until the expression is false

Iteration – do-while loop Example

```
int x = 0;
do {
        System.out.print(x+” “);
        x++;
} while(x<=10);
```

RESULT: 0 1 2 3 4 5 6 7 8 9 10

Diffrence between while and do-while

```
int x = 11;
while (x<=10) {
        System.out.print(x+” ”);
        x++;
}
```

RESULT: NOTHING

```
int x = 0;
do {
        System.out.print(x+” “);
        x++;
} while(x<=10);
```

RESULT: 11

Iteration – For Loop

```
for (statement1;expression;statement2) {
        statement3;
        statement4;
        …
}
```

- statement1 is run once before entering the loop
- loop will run until expression is false
- for each iteration, statement2 is ran

Iteration – For Loop Example

```
for(int x=0;x<=10;x++) {
        System.out.print(x+” “);
}
```

RESULT: 0 1 2 3 4 5 6 7 8 9 10

General Rule for Iteration Statements

- for and while with one statement does not need {}

```
for(int x;x<=10;x++) {
        System.out.print(x+” “);
}
```

is equal to

```
for(int x;x<=10;x++)System.out.print(x+” “);
```

Method

```
public static void main(String[] args) {
        ....
}
```

Scope – public, default, private, protected

Modifier – static, abstract

Return Type – void, int, long, double, String, etc ….

Method Name

Parameter – Type name, EX. int x,String abc,double fp

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

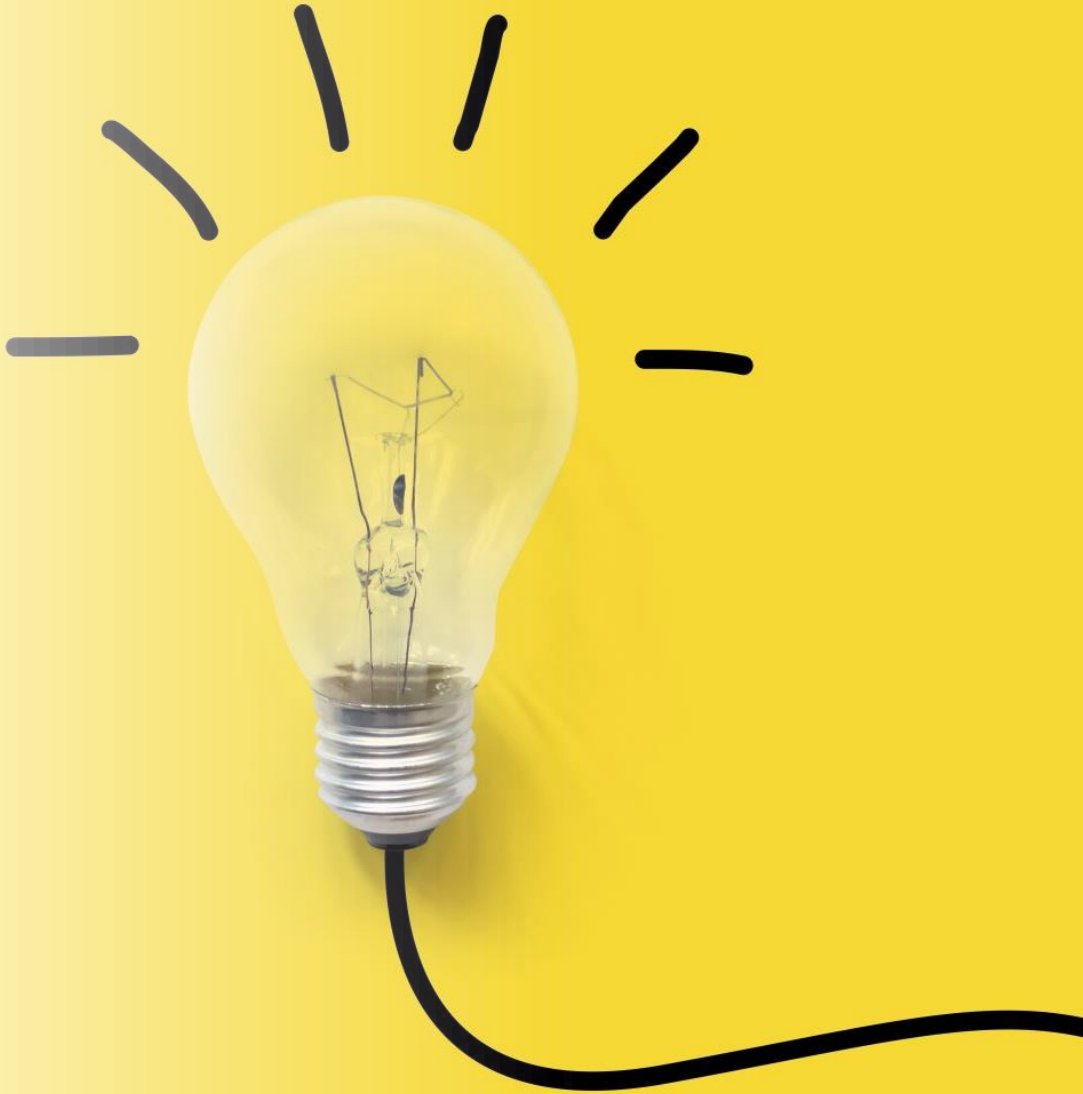
Method – Scope (Explained)

	within non-static method of same class	within static method of same class	within non-static method of other class	within static method of other class
Non-Static Method or Variable	Access directly	Access with object reference	Access with object reference	Access with object reference
Static Method or Variable	Access directly	Access directly	Access with object reference	Access with object reference

Method – Static Modifier (Explained)

Method Overloading

- Method overloading is methods with same name but different input parameter.
- Only one method with matching input parameter will be run.



Method Overloading Example

Without Method Overloading

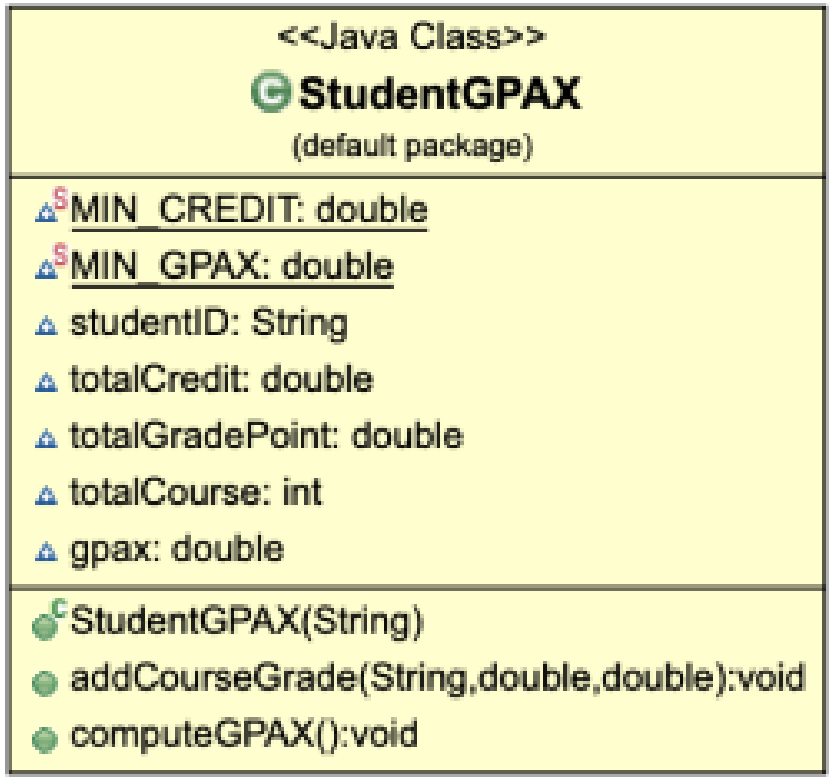
```
int add2(int x, int y)
{
    return(x+y);
}
int add3(int x, int y,int z)
{
    return(x+y+z);
}
int add4(int w, int x,int y, int z)
{
    return(w+x+y+z);
}
```

With Method Overloading

```
int add(int x, int y)
{
    return(x+y);
}
int add(int x, int y,int z)
{
    return(x+y+z);
}
int add(int w, int x,int y, int z)
{
    return(w+x+y+z);
}
```

EX1	add(x,y) is equal to add2(x,y)
EX2	add(x,y,z) is equal to add3(x,y,z)
EX3	add(w,x,y,z) is equal to add4(w,x,y,z)

Creating Object from the given class file



StudentGPAX student;

student = new StudentGPAX("Alex"); //Initialize the object and Run the Constructer method with the parameter String "Alex"

alex.addCourseGrade("Course",10,10);

alex.computeGPAX(); //non-static method must be access through the created object

StudentGPAX.MIN_CREDIT; //Static Field should be accessed staticly.

student.studentID; //non-static field must be accessed through the created object