Saladin Minhaaj
Java Mentoring Session - 01
02/25/2013

What is Java

Java is a software technology which consists of Java programming language and Java software platform/JRE (Java Runtime Environment). It was developed by SUN Microsystems in 1995. James Gosling (a Canadian computer scientist, holds a PhD in Computer Science from Carnegie Mellon, worked at SUN from 1984-2010, and now working at a tech start-up named Liquid Robotics) is known as the Father of Java Programming Language. First version of Java was JDK1.0, and most current is Java SE 7. Java also has 2 other current editions- Java EE for enterprise applications, and Java ME (Micro Edition) for embedded systems e.g. mobile devices.

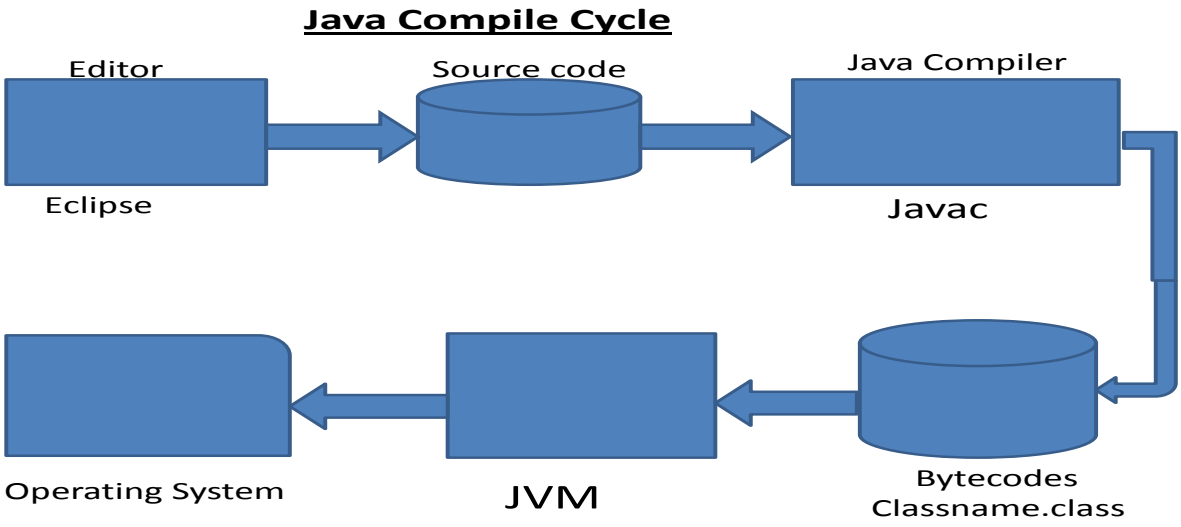
Java language is a high level, object-oriented programming language. Programming languages are generally classified into three levels: high level, mid level, and low level. High level languages are close to human languages e.g. Java, C++, Python etc. Mid level language is the assembly language. Low level language means machine language, e.g. Binary language, JVM bytecode. Languages can also be classified according to the organization of code: Procedural language and Object oriented language. In procedural language, all the codes are written in one single file/class, and in object oriented language, codes are divided into classes.

Java is a platform independent (more specifically, cross-platform) language, which means we can write code in one OS, and easily run in another OS, using JVM. Not all the high level languages have this privilege. This is why SUN used to have a slogan "Write Once, Run Anywhere" (WORA). Moreover, Java applications can be distributed over a network, and it is secure.

Java is both a compiled and interpreted language. Both the terms "Compilation" and "Interpretation" basically mean translation of programming languages. Difference is, in compilation, level of language changes. For example, when we compile a java program, java codes (high level) are translated into bytecode (low level). But in interpretation level doesn't change. When bytecodes go through JVM, it translates bytecode (low level) into binary code (low level).

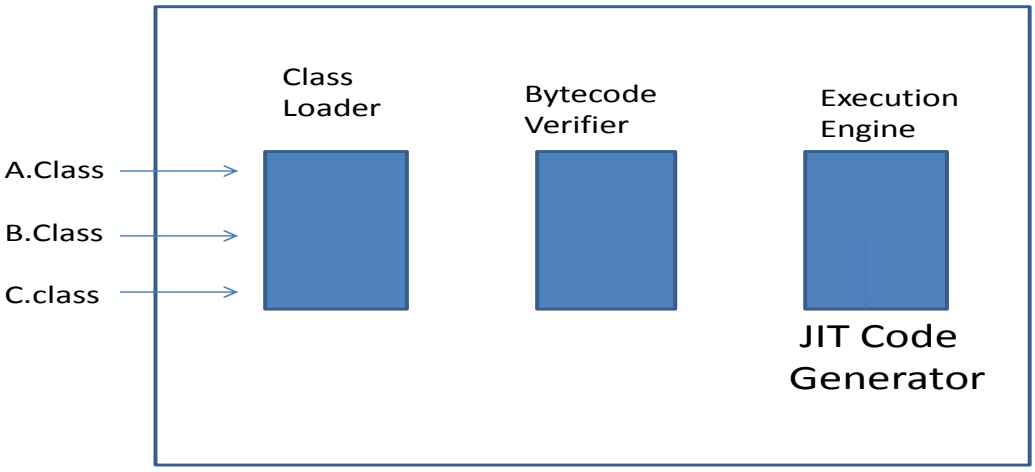
Java platform/JRE has three major components- Java compiler, JVM (Java Virtual Machine) and Java API (Application Programming Interface). Java compiler compiles java source codes. JVM is a virtual computer which runs on top of the OS, and Java API is a collection of around 200 precompiled packages which contains approximately 3800 pre-compiled classes. API is of great importance because Java programmers use the methods and variables of those classes so frequently.

Java Compilation Cycle



Initially, we write java codes in an editor (Notepad) or in an IDE (Eclipse, Netbeans). Our written code is called Source code. Source codes are saved into a Source file, whose extension is .java . Next step is to compile the source code, means translating into bytecode. Bytecode works machine language for JVM (Java Virtual Machine). Extension for bytecode is .class . JVM uses 4 of its major components to run a java program. Class loader loads the bytecode from hard drive into RAM (since JVM resides on RAM). Then the bytecode goes through bytecode verifier. Bytecode verifier verifies that no unnecessary classes can be run on JVM (Java is secured!!) Then bytecodes go to Execution engine, which translates bytecode into native machine code, with The help of JIT Code Generator (JIT stands for Just In Time).

JVM Components



Java Naming Rules & Conventions

Following the rules is mandatory; Eclipse will not allow any name outside the rules.

Java Naming **Rules** (Package, Class, Method, Variable):

1. Can consist of any combination of letters, digits, underscore, dollar sign($)
2. Cannot start with a digit
3. Cannot be a Java keyword

Conventions are standard practices. Before we focus on conventions, we must keep in mind that all the names must be meaningful and short. For example, in our program, we can name the variable which contains the ram size "r", but whoever will read the program will get confused. Therefore, we must name the program components by reflecting their characteristics or functionalities.

A. Package Naming **Conventions**:
   1. Hierarchy:
      nyc.pnt.java
   2. Lower-case letters:
      Avoid this – "NYC.PNT.JAVA"
   3. Must not start with the word "java", this word is only for Java API package names.

B. Class Naming **Conventions**:
   1. Should be noun
   2. First Letter: Capital
   3. If u use 2 or more words, make the first letter of each word –capital

C. Variable Naming **Conventions**:
   1. Should be noun
   2. All lowercase letters
   3. If u use 2 or more words, make the first letter of each word –capital

D. Method Naming **Conventions**:
   1. Should be verb
   2. All lowercase letters
   3. If you use 2 or more words, make the first letter of each word –capital

E. Constant Naming **Conventions**:
   1. Should be noun
   2. All UPPERCASE letters
   3. If you use 2 or more words, separate them by underscores

Starting out with Java

[***Before reading this part, please check the Java source files in Dropbox***]

We will start our Java program by creating a Java project, namely Java_Mentoring_01. There is no rule or conventions for naming the project, but we must keep it meaningful. For the project, I will use a fictional scenario, where I am trying to start my own computer selling business (Saladin Computer Services, LLC) inside PNT Astoria branch. I will only sell HP laptops for the moment.

Next, we will create a package under the project. We will name the package "nyc.ast.pnt". I am in New York City, in Astoria, in PNT. Therefore, this name complies with the hierarchy rule. Then I will create my Computer class. Let's talk about class a little bit.

Class:

Class is the building block for a java program, just like cell in a human body. Several similar classes add up and make a package, same as several cells add up and make an organ in human body. If you break a cell, you will find nucleus, ribosome, mitochondria etc. Likewise, if you break down a class, you will find variables, methods, constructors etc.

Inside a class, there exist 3 major components-

1. Variables: contain data/ information
2. Methods: contain implementation; they are the ones which actually accomplish something. They usually work on/ manipulate the variables
3. Constructors: special type of method. We will discuss about constructors in any of the later sessions.

Let's get back to the program. Inside the Computer class, we will declare 5 variables:

```
int ramSize;
String model;
String processorName = "Pentium 4";
double processorSpeed = 2.13;
char series = 'g';
```

Variable:

Let's talk about variables. In Math, variable means something whose value can change. In Java, it's pretty much the same concept, but from different perspective. In Java, variable means memory location. When we declare ramSize, it takes 4 byte of memory. That memory location is named ramSize, and it will contain any integer type of data. And obviously we can change its value anytime, that's why it's called variable.

ramSize processorSpeed

model series

Variables can be classified as Primitive variable and Reference variable. All the variables we declared are primitive, except String.

Primitive Variable:

Primitive variables are those we can declare using a primitive data type.

There are 8 primitive data types in java:
byte, short, int, long (to hold integer values, differ in memory size, int is mostly used)
float, double (to hold decimal values, differ in size and precision, double is usually used)
char (to hold a single character, such as a letter, digit, and any symbol)
boolean (holds only two values: "true" and "false")

Let's get back to the program again. Now will declare some methods: getRam(), getModel, and printConfig().

```
public int getRam(int ramSize){
         this.ramSize =ramSize;
         return ramSize;
     }

 public String getModel(String model){
          this.model = model;
          return model;
     }

 public void printConfig(){
     System.out.println("This is Saladin Computer Services, LLC");
     System.out.println("This computer's ram size is: " + ramSize + "GB");
     System.out.println("model name: " + model);
     System.out.println("Processor name : " + processorName);
     System.out.println("Processor series: " + series);
     System.out.println("Processor speed: " + processorSpeed);
    }
```

Method:

I have already said that methods perform actions. getRam() method will get the desired ram size from my customer, since I can provide any ram size, and didn't specify this. getModel() method will get the desired model. And printConfig method will print the configuration status of the computer.

All the methods must have a return type or void type. Return type means we are expecting to return any information to the caller. For example, if I tell my younger brother to go to Walgreens, and ask the salesperson the price of a can of Diet Coke (suppose it's $1.09 with tax), and let me know by cell phone, he is supposed to return a double type information to me (the caller). But if I tell him to clean the restroom, I will assume he will do it, and he doesn't need to return any information, so, cleaningRestroom() will be a void type method.

Some methods need parameters. Parameters are any extra information that the method needs to perform its action. For example, after I tell my brother to clean the restroom, he will ask me what he should use to clean-bleach or windex, rag or paper-towel. These are the parameters of cleaningRestroom() method.

The parameter for getRam method is ramSize. Its name is same as the ramSize variable we declared in the beginning of the program.

"this" operator:

To differentiate between the variable ramSize and the parameter ramSize, we use "this" operator. When we say

this.ramSize = ramSize;

Here, this.ramSize indicates the variable ramSize, and ramSize on the right side indicates parameter ramSize. We can avoid using "this" operator by using different names, but in real-world java programs, there are thousands of names we have to deal with. Therefore, we use "this" operator" to reduce the complexity of the program.

One last thing. In the printConfig method, I used System.out.println() method. This is the standard method in java to print something in the console. We will explore more input-output mechanisms later in this course.

Object:

Now let's start the most important concept in Java – "Object". Only because of object, Java is called Object-oriented language. In programming world, an object is a real thing that we can touch/feel. Class is the design/template to create an object. While learning java, always remember this sentence: "Class is the template of an object, and object is an instance of a class". When we wrote Computer class, we just designed how our computer will look like, we haven't made it yet. It's in designing phase.

Now let's create an object- a computer for my first client John. We declare another class inside the same package, named- JohnComp. Inside the class, we start by declaring the main method:

```
Public static void main(String args [] ){
```

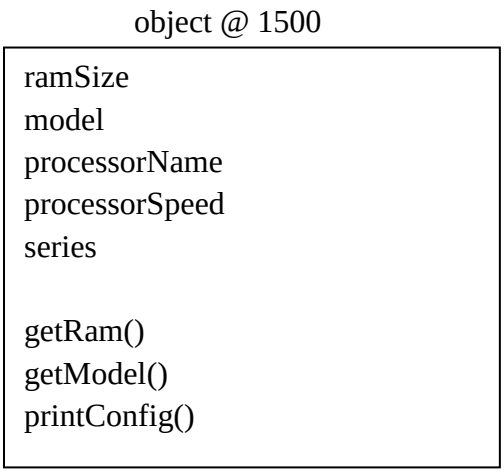
This main method is the starting point of java program. It is a method, which means it accomplishes something. Its function is to provide JVM all the information to run a program. It is the only contact between your code and JVM. When JVM runs the Java program, it first looks for main method, and then executes what the main method implements.

So now, we will declare a java object out of Computer class. The syntax of declaring a Computer object is:

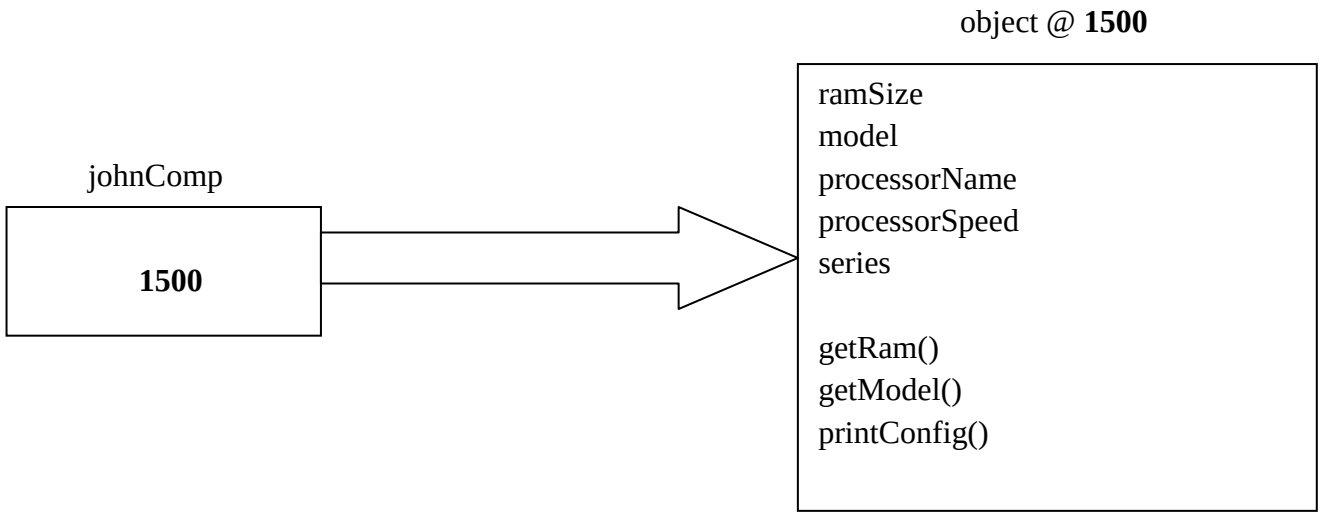
```
new Computer();
```

"new" operator:

When we declare an object, it gets its own copies of variables and access to the methods of that class. Here, "new" is an operator. It dynamically allocates memory space for an object, stores data in that memory space, and returns the address of the memory space. Suppose the memory location of the object is 1500. Here, ramSize specifies the object's copy of the ramSize variable (NOT the original ramSize of the Computer class), and getRam() specifies this method can only work on the copies of the variables of the object.



Since we used the "new" operator, it has already returned the value of the memory address (1500). Now, we need some kind of variable to hold this value. We use Reference variables to hold the values of memory locations. This is why we will declare a reference variable johnComp and assign it to the newly created object. And all the copies of variables and access to the methods will be assigned to johnComp.



Here, johnComp has the address of the memory space, and the memory space has all the variables and methods. We can say that johnComp refers to the actual object. That's how the term "Reference variable" came into existence.

Since object has physical entity, values of all the variables must be specific. While creating an object, it gets its own copies of variables with the values assigned by the template class. If any variable isn't assigned any value in the template class, object's variable copies will automatically be assigned to some default values. I will explain it later in this page. I didn't assign any value to ramSize and model (In this scenario, I gave the customer the opportunity to customize ram size and model). But when I make the physical computer for John, I have to know the ram size and model. Therefore, I have to ask John what size of ram and what model of HP computer he would prefer. This means I have to manipulate the copies of johnComp's ramSize and model. Therefore, I will use the methods that manipulate these two variables, which are getRam() and getModel().

Syntax for calling these methods are:

```
johnComp.getRam(8);
johnComp.getModel("Pavilion");
```

"Calling method" simply means accessing the method. There is another way to assign values to these variables:

```
johnComp.ramSize = 8;
johnComp.model = "Pavilion";
```

Here, we are directly accessing the variables (again!! We are accessing John's copies, NOT the original ones) and assigning values to them.

If we don't call getRam() and getModel() methods to assign values to the variables, nor directly access the variables, what will happen? I have mentioned earlier that Java has a special process of assigning default values to the objects' own copies of variables, if the user doesn't. In case of numerical variables, the default assigned value is 0, for String its "null", for char, it's ' ' (empty space), and for boolean, it's "false". In this program, I assigned the copies of the variables directly:

```
johnComp.ramSize = 8;
johnComp.model = "Pavilion";
```

An object can change its variables' values anytime during the program execution. Suppose John isn't happy with the 'g' series of Pavilion model and the speed of the processor. He wants to switch to 'd' series, and make the speed double. He also wants to see his final configuration status, so we will call printConfig() method. We can write our code accordingly:

```
johnComp.series = 'd';
johnComp.processorSpeed = 4.26;
johnComp.printConfig();
```

After we call the printConfig method, our console will look like following:

```
This is Saladin Computers, LLC
This is your computer
Model is: Pavilion
Series is: d
Ram size is: 8
Processor speed is: 4.26
Thank you !!!
```

So we have successfully sold our first computer to John. Now one of John's friends, Fred, whom we don't know well, lives in Bristow, Virginia. He wants to buy a computer from us, and we are trying to sell him a computer from PNT Virginia branch.

We will declare another package named vir.bri.pnt . Under that package, we will create a class named FredComp. Inside that class, we will declare main method, and inside main method, we will declare another Computer object, and will assign the object's reference to fredComp reference variable. Here, Eclipse will show an error message, that we have to import the nyc.ast.pnt package which contains the Computer class. Importing the package means, whenever we want to create any object out of a class which resides in a different package, we have to import the class first. The syntax is: [package name].[class name]

```
Import nyc.ast.pnt.Computer;
```

Access Specifiers:

Since we don't know Fred that much, we have to restrict his access to our resources. In order to do this, we have to take help from the access specifiers of Java. Access specifiers are Java keywords which we can use in front of any Java component's (Class, Variable, or Method) name to restrict the access to that component. There are 4 access specifiers in Java: public, private, protected, default (no access specifier).

public: Access is open to all. If a class, variable, or method is declared public, it can be accesses from ANYWHERE.

private: Access is open only to the same class. Top-level classes cannot be private, only nested classes can be (since we haven't yet covered the topic of nested class, I'll skip this topic). If a variable is declared as private, when an object gets his own copy of that variable, that object cannot access the variable directly. Then the object can access the variable at first by accessing any method of that class which works on the variable. Likewise, if a method is declared as private, it cannot be accessed by any object. Then the object has to access the variable directly. CONFUSING!!! Isn't it? Let me give an example in the following page.

I have said before that johnComp can assign a value to ramSize variable in two ways.

1. johnComp.ramSize = 8; OR
2. johnComp.getRam(8);

Now, if we declare ramSize as private:

```
Private int ramSize;
```

then we cannot use 1st statement to assign any value to ramSize (assuming that getRam() method is NOT private). We have to use 2nd statement.

On the other hand, if we declare getRam() method a private:

```
public int getRam(int ramSize){
        this.ramSize = ramSize;
        return ramSize;
    }
```

then we cannot use the second statement to assign any value to ramSize, we have to use the 1st statement (assuming that ramSize variable is NOT private).

protected: Access is restricted only inside the package. Protected components behave like public inside the package, and like private outside the package. In our program, we have made FredComp class outside the package. Now if we make ramSize variable as protected, we cannot assign its value in fredComp object directly, we have to use the method. (When we will learn about the concept of Class Extension later in this course, we will explore protected access specifier more).

default: default means no access specifier. Initially when declared our 5 variables, we didn't specify any access specifier, so they have default access specification. Default works the same as protected; there is only one distinction in Class Extension (we will discover later in this course).

“static” keyword:

Now what if Fred is a computer expert and he can make his own computer, but he doesn’t have the series information. He doesn’t know which series he should use- ‘g’ or ‘d’. So he wants to use only the series information of the Computer class. Can we give him the information about computer series without making his computer (without creating fredComp object)? YES, we can, by using “static’” keyword. Let’s talk about “static” keyword.

Generally, static in java means—something that belongs only to the class it is in, not to any instance (object) of the class. You don’t need to create an instance of the class to access the static member. “static” keyword can be described in 4 different contexts:

Static class:

Top-level classes can’t be static. Only nested classes can be static.

Static variables:

- These variables belong to the class and not to the object; a single copy to be shared by all the instances (objects) of the class
- Static variables are initialized only once, at the start of the program execution
- A static variable can be accessed directly by the class name and doesn’t need any object
- Static variables are quite rare

Static constants

Static constants are common. Since we haven’t discussed about Java constants in this session, we will talk about this in any later session.

Static methods

Static methods cannot access non-static variables of the class, but they can access the static variables.

Static methods are methods that do not operate on the copies of object’s variables. For example, the pow method of the Math class (in Java API) is a static method. The expression Math.pow(x, a) computes the power x^a. It does not use any Math object to carry out its task.

Now let’s get back to the program. If we want to give Fred only our series information, we will declare the series variable as static:

```
Public static char series = ‘g’;
```

Then we don’t need to create any Computer object in FredComp class. If Fred wants to access the series and print the value in console, he can use the following statement:

System.out.println(Computer.series);

Console will show the value of the series variable, which is ‘g’.

Comments:

Finally, let's talk about comments. Comments are side-notes. In Java, comments are not mandatory, but they are VERY important for the readability of the program. If I write a program today, and want to update this after a year, obviously I won't remember every single detail, some of which might be crucial for program execution. If I write some comments after those statements which need some explanation, then I don't need to remember their descriptions. When javac compiles Java program, it ignores the comments, so comments don't affect the program.

Comments are 3 types: single-line, multiple line, multiple line for javadoc. We don't need to learn javadoc right now. We will be fine just using first 2.

Single-line comment: //
Multiple-line comment: /*
*
*
*
*/

We need to choose carefully what we will write in comments. When we declared "series" variable, we can write comments this way:

char series = 'g'; // We declared a char type variable named series and assigned a value 'g' to it

This comment is useless. Anyone can see what we have done here. We would rather use this comment:

char series = 'g'; // series holds the value for the HP Pavilion series that we are currently offering

In one word, it's smart to comment WHY we are doing something, rather commenting WHAT we are doing.

****

This is all about my 1st mentoring session. I hope this will be beneficial for you guys. Any question or comment or concern would be greatly appreciated. See you next week. Adiós !!!!!!!

Saladin Minhaaj
(646) 420-2509
akms.minhaaj@gmail.com