LowLevelLanguage/MachineLevelLanguage- Machine can undestand only binary language. It is in binary format. In the form of 0's and 1's

AssemblyLanguage/MidLevelLanguage- Intermediate language which looks like processsor instructions/cmds which is not completely in binary

and also not completely human readable.

HighLevelLanguages - Human Readable or more like general english.

int a=5;

int b=10;

int result = a+b;

Compiler : Compiler is a program that converts human-readable code to machine-readable instructions.

eg for compiled languages C, C++, Objective-C, Scala

Interpreter : Interpreter convert the humanreadable code to machine code while executing the program line by line.

eg for interpreted languages:

Python,Ruby, JavaScript,

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

COMPILED LANGUAGES

are written in a code that can be executed directly on a computer’s processor. This is because a compiler has translated the code into the computer’s “native” language up front, well before the program is even run. This process can take many passes before it is optimized as machine code, but the output is always code that’s ready to be executed—and that executes efficiently, as a result.

A compiler is a program that converts human-readable code into computer-readable instructions—a process that only happens once in the lifespan of that code. Initially, it takes a bit longer because the compiler has to rearrange, optimize, or “compile” object code first.

Once a compiler takes in source code, optimizes it, and then generates the object code, another step has to happen. Object code can’t always run on its own; it needs dynamic load libraries and other bits of code—code that’s unique to the operating system, which is housed in those libraries. Within the compiling process, these libraries are linked to object code in a “linker,” a part of the compiler that bundles those bits of code together, then it’s good to go.

That final product—the object code packaged up with the libraries—is the compiling program’s executable file—a .exe file.

Some compiled languages include:

• C

• C++

• C#

• Erlang

• Objective-C

• Pascal

• Scala

• Swift

• Smalltalk

• TypeScript

INTERPRETED LANGUAGES

An interpreted language is any programming language that isn’t already in “machine code” prior to runtime. Unlike compiled languages, an interpreted language’s translation doesn’t happen beforehand. Translation occurs at the same time as the program is being executed.

Many of an interpreted language’s instructions can be executed directly, without compiling to machine code; however, when certain code is required, an interpreter steps in during runtime and translates it on the spot.

Interpreted languages have a major advantage: they’re portable, which means they can run on different operating systems and platforms. Also, because they’re translated on the spot, they’re going to be optimized for the system on which they’re being run. That means there are no middle steps, less memory space is required for interim object code, and there’s no need to worry about platform-specific code.

How interpreting works

The interpreter is a program that converts source code—the human-readable code mentioned above—into machine code each time you run the program, one line at a time. It starts interpreting each instruction immediately upon execution, which means that the resulting program runs slower than a compiled program—it’s got more going on at runtime. Compiled languages, on the other hand, have already been through this translation before program execution, so they’re arguably faster.

Interpreters have some other bonuses, too. They’re especially helpful for reviewing, running, and testing an application’s functionality during development because they’re able to execute high-level programs immediately—and generate helpful error reports. Also, they allow programmers to make small, step-by-step changes during the development process, incrementally, which complements a step-by-step process for adding and then testing smaller sections of an application.

Some interpreted languages include:

• Java

• JavaScript

• PHP

• Perl

• Python

• Ruby

Simula was the first object-oriented programming language. Java, Python, C++, Visual Basic .NET and Ruby are the most popular OOP languages today. The Java programming language is designed especially for use in distributed applications on corporate networks and the Internet. Ruby is used in many Web applications. Curl, Smalltalk, Delphi and Eiffel are also examples of object-oriented programming languages.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Java- compiled and interpreted.

Java is an Object Oriented programming lanaguage.

It has Classes and Objects.

Mobile class - Class defines model or blueprint for object. It has properties and methods.

Object is that which exist or it is instance of class.It has state(properties) and behaviour(action/methods in which we define logic).

Samsung, Apple, GoogleNexus - has color, manufacturer, ram features, camera.

Object Oriented Programming Concepts / OOPS:

Inheritance

Polymorphism

Abstraction

Encapsulation

Java - It is platform independent language.Write Once Run Anywhre (WORA).

Java is a language and platform.

I have written a program in java and compiled it in windows OS, I can still execute this program in MAC OS.

Features of Java:

Simple- It eliminates pointers which are used in c and c++ for memory allocation and deallocation. Java memeory managemenr is taken care by JVM.

ObjectOriented- In Java everything is objects and classes

Distributed- It works with various systems on n/w and can handle any protocols(Tcp/ip,udp etc)

Robust- It is strong bcz it do not crash easily..Java handles error/exception in program using exception handling features.

Secure- It supports encryption and decryption mechanisms.

SystemIndependent- Java byte code can be executed on any processor and OS.

Portable

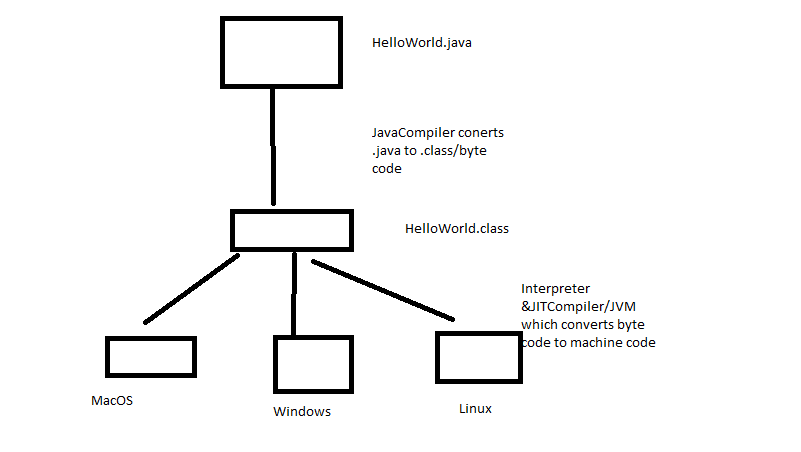
Interpreted- Hello.java--> Hello.class(bytecode)-->machine readable code is done by interpreter

Good Performance- Bcz it handles complex logics execution using JIT compiler - JustInTimeCompiler

MultiThreaded- Java supports threads which help logic to execute parallely

Scalabilty- It can connect to DB, webservices and any remote calls using its API packages.

Dynamic- Make dynamic interactions possible on web apps.



Setup java:

Download Java as per your OS from - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Run exe file. You wil Java folder in C:ProgramFiles

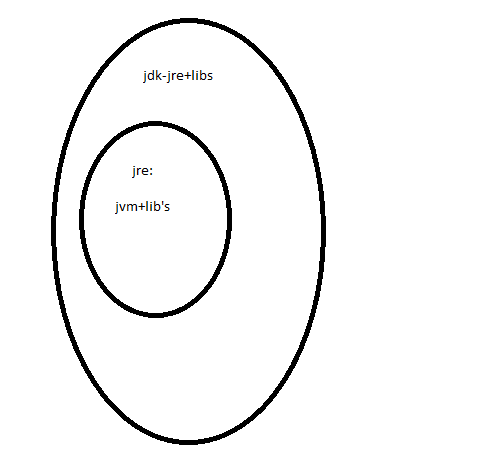
Setup Environment Variables - Add new variable - JAVA\_HOME - C:\Program Files (x86)\Java\jdk1.8.0\_101

and path variable edit and append- %JAVA\_HOME%\bin or C:\Program Files (x86)\Java\jdk1.8.0\_101\bin

JDK - Java developement Kit - set of Java Libraries and JRE - required to develop a program

JRE - Java Runtime Environment - Few libraries (Which has subset of certain libraries required to execute the program) and JVM - executes program

JVM - JavaVirtualMachine- Interprets the program



Java is system independent but jvm is system dependent.

GarbageCollection- memory management section in jvm - this will deallocate memory after program execution.

dll and exe:

dll- dynamic link mibrary- which are used by exe file in their execution

exe- they are executable on machine.

JavaVersions-1 to 1.4, 1.5, 1.6, 1.7, 1.8

|  |  |
| --- | --- |
| **Release** | **Year** |
| JDK Beta | 1994 |
| JDK 1.0 | 1996 |
| JDK 1.1 | 1997 |
| J2SE 1.2 | 1998 |
| J2SE 1.3 | 2000 |
| J2SE 1.4 | 2002 |
| J2SE 5.0 | 2005 |
| Java SE 6 | 2006 |
| Java SE 7 | 2011 |
| Java SE 8 | 2014 |

IDE's - eclipse, intellijidea, NetBeans - Integrated dev environment

**Package** – package is the namespace/folder structure that organizes the set of classes and interfaces.

**Class** : Model or blueprint which has properties and methods.

Members of class are variables and methods.

**Variables** are properties which define state of object:

Syntax- accesModifier Datatype varName ;

**public** String breed;

Variables at class level are called global variables or instance variables and they can be referred in any of the methods of class.

**Methods** are actions which define logic of object.

Syntax- accesModifier returnType methodName(input parameters){

Code or method logic

}

**public** **void** eat(){

System.***out***.println("It eats food");

}

**Object**- instance/representation of class.This has properties and it can call class methods.

Syntax- ClassName objName = new ClassName();

Animal animalObj = **new** Animal();

**NamingConventions:**

Java is caseSensitive.

packageNames – lower case

com.wbl.java

com.wbl.test

variableNames, methodNames – camelCase

display

displayDetails

classNames, interfaceNames – initCaps

Student

StudentDetails

Constants- allcaps

PI=3.14

**Rules for Identifiers/Names:**

Identifiers can have alphabets, numbers, $,\_

Do not start name with number, it can start with alphabets,$ or \_ and it can include numbers.

Each identifier must have atleast one character.

**AccessModifiers:**

Public – can be accessed from anywhere.

Private- can be accessed only **within the class**

Protected- can be accessed within same package **and subclass of diff package.**

Default- can be accessed within the **same package.**

**JavaComments:**

//- single line comments

/\*…\*/- multi line comments

**Constructor in java:**

Constructor is similar to method syntax wise but it is called only **at the time of creation of object.We can make any intialisations using constructions.**

It wil have same name as class name.

It will not have return type.

Its added by default by java if you don’t create one.

It can take inout parameters like method.

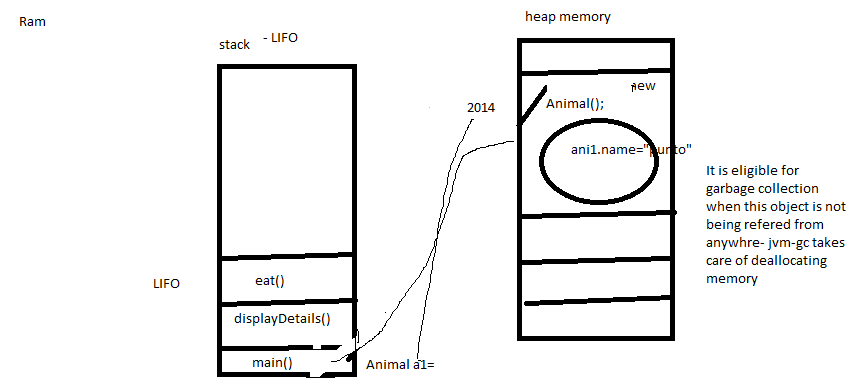
A class can have multiple constructors which is called constructor overloading.

Overloading means you will have same constructor name and different parameters within same class.

**Stack and heap in Java:**

Method execution are done from stack- stack follows LIFO principle.

Objects memory allocation is done in heap

****

primitiveDatatypes:

keywords in java:

this

static

final

Create a class and define properties and methods.

Create Objects for class and access properties and methods.

**OOPS**:

**Encapsulation**: Binding the variables and methods together , hide implementation behind an interface.

Encapsulated code will have 2 things:

Instance variables are protected(we can use private modifier and protect it)

We provide public getters and setters to access the instance variables.

**Inheritance**:

Inheritance allows a class to be subclass(child class)of a superclass(parent class) – therefore subclasses inherit public and protected variables and methods of the superclass.

Use: Avoid Repeating the logic or redundancy of code – Making the code reusable.

Car- Benz is a car, BMW is a car

Car{

Public void doARide(){

Print(“evry car can do a ride”);

}

Benz extends Car(){

Public void setBenzFeatures(){

Print(“luxury car”);

}

}

BMW extends Car{

}

Polo extends WV{

}

Animal – Dog is a animal, Cat is animal

Class Animal{

Public void commonFeatures(){

Print(it has four legs)

}

Public void setSound(){

Print(“they make sounds”)

}

}

Dog extends Animal{

Public void setSound(){

Print(Dog says bow)}

}

Cat extends Animal{

Public void setSound(){

Print(Cat says meow)

}

}

**MethodOverriding: Overriding only occurs with inheritance.**

The child class methods should have exactly same syntax as parent class - same name, same return type.

You cannot have more restrictive access modifier or you cannot narrow the scope.

You can have less restrictive access modifier or widen the scope.

Lifet🡪right – narrow-wide or morerestrictive->lessRestrictive

Private->default->protected->public

Exceptions

Child class should have same return type as parent but Covariant return type is allowed.

A parent class reference can hold child class objects.

Eg: all below are valid

Animal dog = new Dog();

Animal cat = new Cat();

Dog dog = new Dog();

Cat cat = new Cat();

In case of overriding - the call of the method whether it should call parent class method or child class method depends on which object is getting created at run time.This is called **java virtual method invocation.**

**super:** is a keyword by which we can access parent class members(methods,variables,constructors) in child class.

It is used when you want to refer parent class methods/variables in child

When both parent and child have similar variables/methods.

**this:**  keyword in java – this is used to refer the member(instance variables,methods) of current object from method or constructor.

It can be used inside a method or a constructor of the class.

this is a reference to current object of execution.

**Polymorphism:**

**Abstraction:**

Annotations – that which give certain functionality in easier way

Write a class that describes few fields and methods w.r.to Training.

Write subclasses for QA and UI training with specific implementation for recordings and

course content.