Java

Computer:

Advantage:

- Large Data
- Faster
- Accurate

Human Beings Disadvantage

- Slow
- cannot store or remember large data
- Accurate
- Tiered
- Bored

Language:

HB: K,E,H,Fre,Mal

Computer:

Machine Lelvel Language[Binary Language]

History

Java, having been developed in 1991, is a relatively new programming language. At that time, **James Gosling** from Sun Microsystems and his team began designing the first version of Java aimed at programming home appliances which are controlled by a wide variety of computer processors.

Gosling's new language needed to be accessible by a variety of computer processors. In 1994, he realized that such a language would be ideal for use with web browsers and Java's connection to the internet began. In 1995, Netscape Incorporated released its latest version of the Netscape browser which was capable of running Java programs.

Why is it called Java? It is customary for the creator of a programming language to name the language anything he/she chooses. The original name of this language was Oak, until it was discovered that a programming language already existed that was named Oak. As the story goes, after many hours of trying to come up with a new name, the development team went out for coffee and the name Java was born.

While Java is viewed as a programming language to design applications for the Internet, it is in reality a general all purpose language which can be used independent of the Internet.

While Java is viewed as a programming language to design applications for the Internet, it is in reality a general all purpose language which can be used independent of the Internet.

Compilation:-

1)converting form high level language to machine level language

2)process of converting source code to machine code

3)process of converting source code to object code

debugging:

correcting the errors

Buzzwords/Features Of Java

1.Simple
2.Secure
3.Portable
4.Object-Oriented
5.Robust
6.Multithreaded
7.Architecture neutral
8.Interpreted
9.High Performance
10.Distributed
11.Dynamic

Simple:
•Java easy to learn
•It is easy to write programs using Java
•Expressiveness is more in Java.
•Most of the complex or confusing features in C++ are removed in Java like pointers etc..

Secure:
•Java provides data security through encapsulation.
•Also we can write applets in Java which provides security.
•An applet is a small program which can be downloaded from one computer to another automatically.

•There is no need to worry about applets accessing the system resources which may compromise security.
•Applets are run within the JVM which protects from unauthorized or illegal access to system resources.

Portable:
•Applications written using Java are portable in the sense that they can be executed on any kind of computer containing any CPU or any operating system.
•When an application written in Java is compiled, it generates an intermediate code file called as “bytecode”.
•Bytecode helps Java to achieve portability.
•This bytecode can be taken to any computer and executed directly.

Object - Oriented:
•Java follows object oriented model.
•So, it supports all the features of object oriented model like:
Encapsulation
Inheritance
Polymorphism
Abstraction

Robust:
•A program or an application is said to be robust(reliable) when it is able to give some response in any kind of context.
•Java’s features help to make the programs robust. Some of those features are:
1. Type checking
2. Exception handling

Multithreaded:
•Java supports multithreading which is not supported by C and C++.
•A thread is a light weight process.
•Multithreading increases CPU efficiency.
•A program can be divided into several threads and each thread can be executed concurrently or in parallel with the other threads.
•Real world example for multithreading is computer. While we are listening to music, at the same time we can write in a word document or play a game.

Architecture - Neutral:
•Bytecode helps Java to achieve portability.
•Bytecode can be executed on computers having any kind of operating system or any kind of CPU.
•Since Java applications can run on any kind of CPU, Java is architecture – neutral.

Interpreted and High Performance:
•In Java 1.0 version there is an interpreter for executing the bytecode. As interpreter is quite slow when compared to a compiler, java programs used to execute slowly.
•After Java 1.0 version the interpreter was replaced with JIT(Just-In-Time) compiler.
•JIT compiler uses Sun Microsystem's Hot Spot technology.
•JIT compiler converts the byte code into machine code piece by piece and caches them for future use.
•This enhances the program performance means it executes rapidly.

Distributed:
•Java supports distributed computation using Remote Method Invocation

(RMI) concept.
•The server and client(s) can communicate with another and the computations can be divided among several computers which makes the programs to execute rapidly.
•In distributed systems, resources are shared.

Dynamic:
•The Java Virtual Machine(JVM) maintains a lot of runtime information about the program and the objects in the program.
•Libraries are dynamically linked during runtime.
•So, even if you make dynamic changes to pieces of code, the program is not affected.

Object Oriented Programming

OOPs[object oriented programming concepts]

---

class:-

it is a design of an object

it is a blue print of an object

it is a user defined data type[programming]

Object:-

it is the instance of a class

it is the run time entity of class

it is an variable of a class[programming]

OOP pillars

---

1)Encapsulation

2)Abstraction

3)Polymorphism

4)Inheritance

Encapsulation:-

a)it is a process of data hiding

b)it is a process of binding data and functionality together

Abstraction:-

a)it is a process of accessing hidden data indirectly

b)it is a process of hiding unimportant data or functionality.

Polymorphism:-

many forms of functionality

Inheritance

taking properties from parent class

properties of an object

---

1)state

2)behavior

3)identity

Object oriented Programming stages: [civil]

---

1)analysis phase

2)design phase

3)implementation phase

Hello world Program

Program:

```
class A {
public static void main(String v[]) {
System.out.println("hello world");
}}
```


save:

filename.java

compile:

javac filename.java

interpret:

java Classname

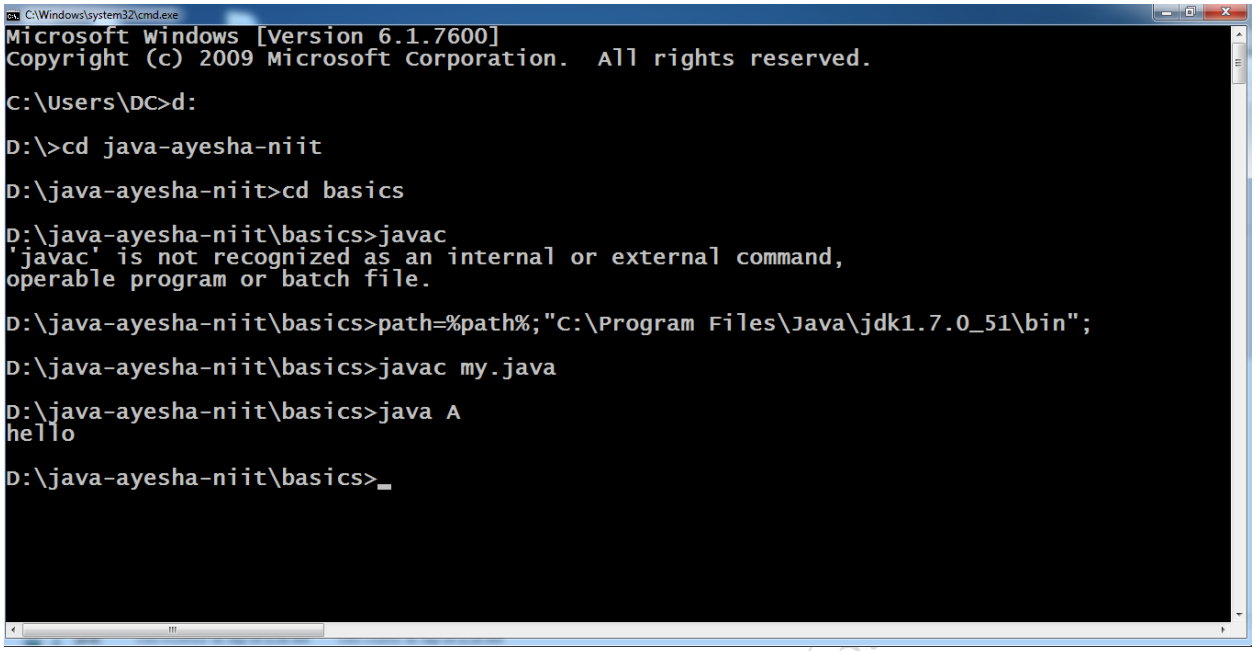
path setting for java compiler

---

path=%path%;"C:\Program Files\Java\jdk1.7.0_51\bin";

or

path="C:\Program Files\Java\jdk1.7.0_51\bin";%path%;



Explanation

A

class_name

why main is static in java

main exists even if object is not created

static members exists even if object is not created

System

it is a classof package "java.lang"

out

built in object of "PrintStream" class of package "java.io"

println

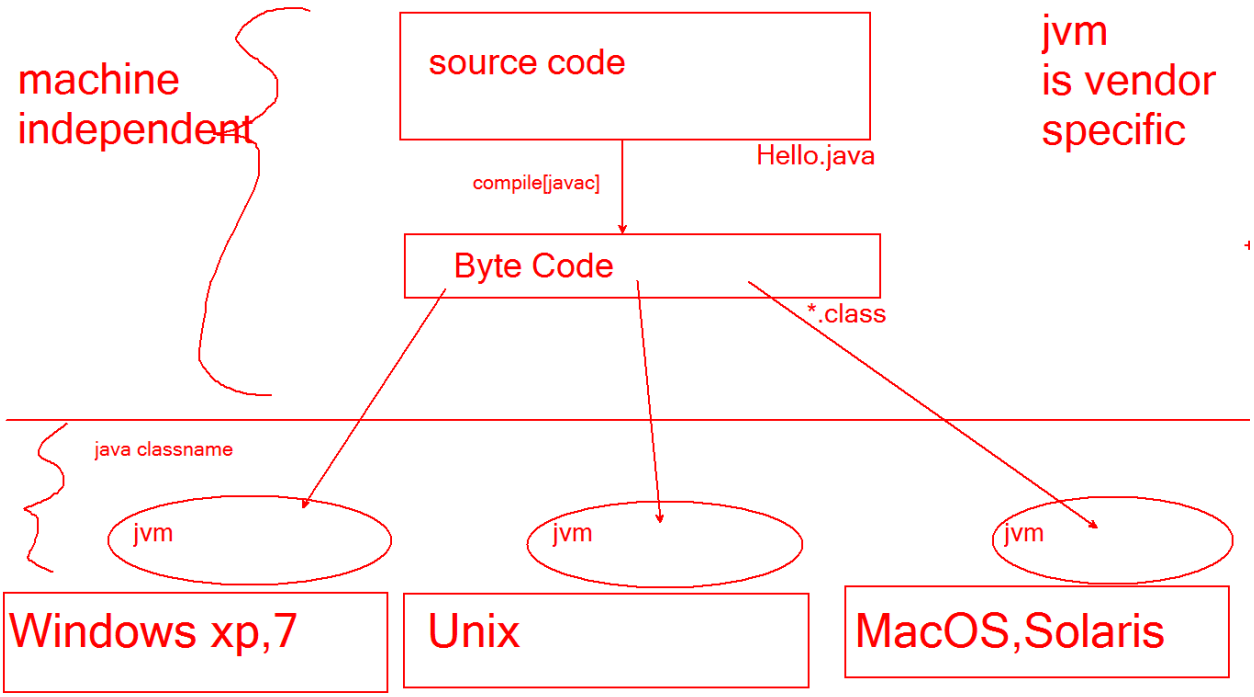
it is a function of "PrintStream" class of package "java.io"

print

it is a function of "PrintStream" class of package "java.io"

printf

it is a function of "PrintStream" class of package "java.io"



Object Oriented Programs

Basic syntax of class in java:

---

class class-name

{

data members/member variables

member functions

}

Example:

class Ex

{

int num1,

String v2;

float x;

void set()

{

```
}
void display()
{
}

}
```

class-name: Ex

members: num1,v2,x,set,display

member variables: num1,v2,x

member fucntions: set,display

Note:

1)Every class in java must be named according to naming rules

2)Every class in java is a child/sub class of "Object" class

3)"Object" class is the mother of all classes in java

Naming Rules of Java Identifiers[variable,function,class etc..]

---

1)Start the name with alphabet,underscore or $

2)A name can consists of alphabet,numbers,underscore and $

3) Dont use java language keywords

There are 50 keywords in java

All keywords in java are in lower case

java is case sensitive language

Note:

Every class must have a name

Every class in java is child class or sub class or "Object" class

Every java file has default inport of "java.lang" package

"Object" class is mother class in java

Naming conventions

---

1)class:

Noun

upper case at begining of each noun

dont use underscore,$,numbers

2)objects:

meaning

completly in lower case

3)methods:

verb[run,prit,write]

first name lower case 2nd onwards 1st letter upper case

example:setName,getName,printNVvalue

dont use underscore

Example:

print()

printMyName()

4)package names:

completely in lower case

don't start from java

5)constants:

completely in upper case separated by under score

Example: SIZE,MAX_VALUE

Polymorphism

Polymorphism:

Overloading

Defining More than one function with same name but with different function signature

i.e, different parameter types

a)function overloading

b)constructor overloading

c)operator overloading[not in java,supported in c++,c#]

Example:

```
void add();   //0
void add(int a);      // int
int add(int a,int b); //1  int,int
```

Function:

Name

Parameters/Arguments[fixed]

count,data type used,sequence

Return type

function signature:

Parameters/Arguements[fixed]

count,data type used,sequence


```
void add(int a,float b)              //2    int,float
{
}

int add(float a)              //1      float
{
}
```

```
void fun1(){}
void fun1(int a){}
void fun1(float a){}
void fun2(int a,int b){}
void fun2(int a,int b,int c){}

void fun1(int a,int b){}    //2  int,int
int  fun1(int x,int y){}     //2 int,int
```



```
class A
{
void fun1()
{
}
void fun1(int a,int b){}
}
```

```
class B
{
void fun1(int a)
{
}
void fun1(float x){}
}
```

```
void fun1()  //0
{}
void fun1(int a)     //1  int
{}


void fun2()  //0
{
}
int fun2(char a)     //1  char
{
}
void fun2(int a)     //1  int
{}
```

```
void funx(int a,int b)                //2 int
{
}
void funx(char ,char b)    //2 char
{}
void funx(int a,char b)     //2 int,char
{}
void funx(char a,intb)      //2 char,int
{}
void funx(int a,int b,int c)          //3 int
{}


class A
{
public:
A()
{
}
A(int a)
{
}
```

};

Error in overloading:

void add();

int add();

same name with same parameters

even if return type is different it cannot be considered

void add(int a,int b);

2 int,int

int add(int x,int y);

2 int,int

same name same function signature

Inheritance

Inheritance

reusable

modified

types of relationship between classes

a) is-a relationship [inheritance relationship]

```
class Bird
{
}
class Parrot extends Bird
{
}
```


b) has-a relationship [composition relationship]

```
class A
{

class B
{
}
```

}

Car has-a Engine

types of inheritance

1)single level inheritance

2)multi level inheritance

3)hierarchial inheritance[class having more than one child class]

4)multiple inheritance[class having more than one parent class]

language	parent class	child class
java	super	sub
c#	base	derived/sub
c++	base	derived/sub

```
class A
{
}

class B extends A
```

```
{
}

class C extends A
{
}

class D extends B
{
}
```


overriding

---

overriding means redifining the inherited methods with same signature and same return type

Note:

Access-Modifiers can be:

private-->private,no-modifier,protected,public

no-modifier-->no-modifier,protected,public

protected->protected,public

public-->public

keywords[super,final,abstract,interface]

Final KeyWord in Inheritance

---

1)Final keyword is used to class for avoiding inheitance

if class is final it cannot be inheirted

2)Final keyword is used to method to avoid overriding

3)Final keyword is used to variables to create constants

super base[c#]

---

1)To call parental constructors respectively if overloaded

2)To refer parental member variables

3)To call parental methods if overridden in child

Note:

1)order of constructor calling:

constructors are called from parent to child class

2)super statement must be first and only one statement in child constructor

Abstract Class

---

1)a class is called as abstract class if it is declared as abstract

or

class is declared as abstract if it consists of atleast one abstract method

2)a method is called as abstract if it is declared as abstract method

method is declared as abstract if it is not defined

Note:

**)method without a body is called as abstract method

1)Abstract class Cannot be instatiated

2)parental object can be instatiated to child class

3)constructors cannot be static in java

4)constructors cannot be overidden from child class

5)static members are not at all inherited so no overriding

6)abstract method cannot have a body

****)

private final methods can be overridden

illegal combination

private abstract

final abstract

abstract static

Interfaces:

interfaces are just like classes in java

interface can be empty

interface can have fields/variables but with value only

interface can have methods which are abstract only

interfaces cannot be instantiated

interface can have child interfaces

interface can have child classes

Note:

interface cannot be a child of class

member variables of an interface are public static and final by default

member functions of an interface are public and abstract by default

interface will not have constructors or it cannot be defined also

child cannot reject or select parental members while inheriting

constructors and static members are not inherited at all

constructors cannot be static in java,c++

Contact:

Chandra Prasad R

9686856330,7204529777

raju.chandra23@gmail.com