Core Java

Some important points to remember

Here I am giving some important points on Core Java from Mr. Suresh's core java notes – 2001.

Its just collection of some points but not the entire notes on Core Java and I am presenting this notes by thinking that you know Core Java well (at least basics and important concepts)

Note: This is not written,edited like a text book on JAVA. iNetSolvians has prepared this book by taking the notes in the class.

My sincere thanks to all students and faculty who has typed and reviewed this material.

Madhav L
(www.geocities.com/megamadhu
http://free.7host06.com/lmadhav)

Contents:

OVERVIEW OF C++:
C++ was invented by "Bjarne Stroustrup" in 1979, Bell Laboratories, Murray Hill,New Jersey. Stroustrup initially called the new language C with classes. In 1983, the name was changed to C++, C++ extends C by adding object-oriented features. C++ was standardized in November 1997, and an ANSI/ISO standards for C++ is now available.

NEED OF C++:
The use of structured programming language (like C) enabled the programmers to write, for the first time, moderately complex programs fairly easily. However, even with structured programming methods, once a project reaches a certain size, its complexity exceeds what a programmer can manage. By the early 1980's many projects were pushing the structured approach past its limits. To solve this problem, a new way to program was invented, called Object Oriented Programming (OOP). OOP is a programming methodology that helps organize complex programs through the use of Inheritance, Encapsulation and Polymorphism.

INHERITENCE: It is the process by which one object acquires the properties of another object.

ENCAPSULATION: It is the mechanism of combining/binding the code and data it manipulates and keeps both safe from outside interface and misuse.

POLYMORPHISM: One interface, multiple methods. This means that it is possible to design a generic interface to a group of related activities. This will reduce the complexity by allowing the same interface to be used to specify a general class of action. Compiler will take care of selecting the specific action (i.e. method) to each situation. Programmer does not need to make this selection manually combining these three, to produce robust and scalable programs than does the process-oriented mode.

'Java' is an Object-Oriented Programming language.

Variable: It is named memory location (that may be assigned a value by your program) that may contain a value that can be changed during the execution of a program.

A SIMPLE JAVA PROGRAM: Example.java

/*This is a simple java program
To compile: “javac Example.java"
To run: “java Example */

```
class Example{
//program begins with a call to main()
public static void main(String args[ ])
{
System.out.println("hello world");
}
}
```

Note: In Java source file is officially called a 'compilation unit'. This is a text file that contains one or more class definitions.
In Java all the code must reside in a class. By convention, the name of that class

should match with the name of the file that holds the program. Java is case-sensitive.

Compiling the program:
c:\>javac Example.java
Java compiler creates the file called Example.class that contains the byte code version of the program.

To run the program: You must use java interpreter, called java. Java byte code is the intermediate representation of your program that contains the instructions the java compiler will execute

c:\>java Example (output-Result of run)
c:\>java -p Example (to see the byte code)

In java, all program activity occurs within xxxe. This is one reason why all java programs are (at least a little bit) object oriented. All java applications begin execution by calling main().

args[]: Receives any command line arguments present when the program is executed.

String: Stores character string.
System.out.println("---------");

System: Is a predefined class that provides to the access to the system

out: Output stream that is connected to the console.

println: Displays the string which is passed to it(method).

void: Simply tells the compiler that main() does not return a value.

Very important concepts in developing the applications are:
Operating system
Networking technologies
Data structures
Compiler constructions
Language designs

TCO: Total cost of ownership is also considered, while developing the application.
Delphi developed by Borland is 20% faster than VB. but that is failed because of the less marketing strategies.
Generally 50% of the features are not useful to the customers, using any software, but to capture as many no of customers, all the features are introduced by the s/w developing companies.

WHILE DESIGNING AN APPLICATION THESE POINTS ARE IMPORTANT:-
Creating the extensions to the product.
Feature extendibility of the product and it is very easy manner to do this.
Lowering the total cost in extending the product.
Product features can be extended by the third-party.
Extending the features of a product by third-party without touching the source code.

Building the s/w (compiling + linking = building) in multiple pieces, so that any bugs in any of the piece, we can modify that particular piece very easily, without damaging the entire s/w.

In Linux Kernel performance level is good at some critical points.

Customer satisfaction is very important in designing the product.

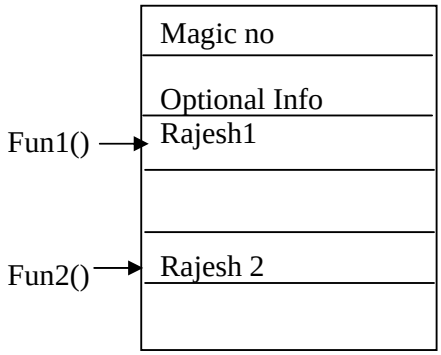
Maintaining the Source in UNIX: Source must be written in n number of files instead of one single file

```
Main()
{
   xyz();
   abc();
   cbj();
}
```

```
Abc()
{
…..
……
…. }
```

Object file contains m/c level language instructions and some information.(functions)
Format used in UNIX is COF format (Common Object File Format)

COF format must consist of a magic number(tradition in Unix community)



Optional Information: Information depends upon the operating system. Here multiple

numbers of sections are available.

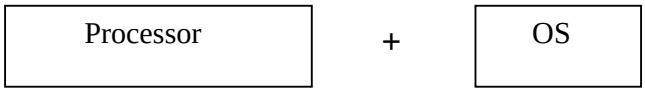
Protocol: Set of standards that are used to communicate between two entities.

Processors:

Not all the CPUs are going to support the same instructions. Every CPU has its own instruction set. Instruction codes for the different CPUs are different, even though they support all the operations.

You need to create separate exe files for different processors. Using the existing instructions, simulate the instructions that we need.

Platform: In general, on which we are standing or sitting. Operating System (OS) on which our program is running.



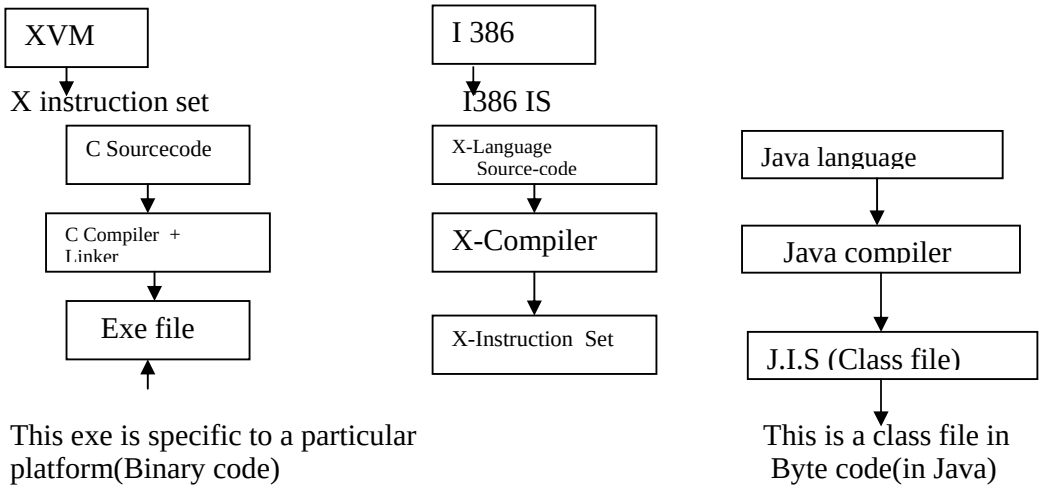
Instructions and Information is different for different type of OS.

Program written in 'C' language using Intel 386 on windows, should not be executed on the other o/s. Execution produced by Windows o/s is portable executable (P.E). In Linux o/s, it is in ELF.

When you have additional s/w at the o/s level and underlying hardware it is possible to run the windows executable on any other Platforms.
Ex: wine tool using this tool we can run the Windows executable on Linux.

Virtual Machine: It is a m/c which is not really exists.
Ex: Like Intel 386, SPARC, and Alpha processors, but these are really exists.

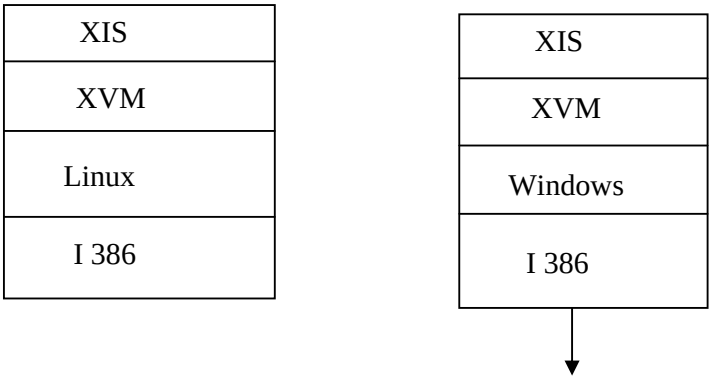
Windows supports portable exe format.
Linux supports ELF.



This exe is specific to a particular platform(Binary code)

This is a class file in Byte code(in Java)

The purpose of the wine tool is to simulate the windows exe on Linux Platform.



Advantage: We can run the final universal format exe File in future platforms without any Modifications using XVM, which is going to simulate the real o/c

JVM:
Java Virtual Machine is a piece of s/w that runs your class file in which they stores the o/p. This will convert the byte code into the m/c dependent code. JVM is to simulate the concept.

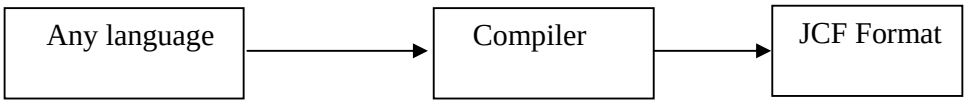
Why JVM stores the class file in a byte format?
The length of all the instructions in JVM are 8 bits (or) 1 Byte .Because of this the JVM stores the class file in a Byte format.
Java language specifications are open to all. anybody can develop the language and if anybody wants to make any changes to the application can be done. Because of this java is popular.
Simulate means pretended to be, have, or feel, reproduce the conditions of.

Java is successful in the market because of the
Advancements of the hardware.
Developing and distribution of files are very easy. So,if any bugs occurs, that particular piece should be modified and send it back to the customers. and downloading of that particular piece is very easy to the customer instead of re-building the entire application.

JCF: java class file format is java Platform dependent.

Is it possible to use c++ language or any other languages to generate JCF?

Yes. Compiler takes the instructions and generates the output according to the way in which you want. to generate JCF we can use any language, but general choice is java.
Ex: smalltalk



Can I generate the output in portal executable or elf format without using JVM?
Yes, but we can't run this exe on the other platforms.

Ex:- schematic allows you to do this, for windows platform.

Tools required to the developers to develop a java project:-

1. **JDK** - Java Development Kit.
To develop java program to convert into java byte code.

2. **JRE** - Java Run-time Environment.
(JVM + set of Libraries) To run java byte code.
According to Javasoft Libraries are called as packages.

Customer need not required to install JDK, only JRE is enough to run the Java byte code.
rt.jar-> Java archive.

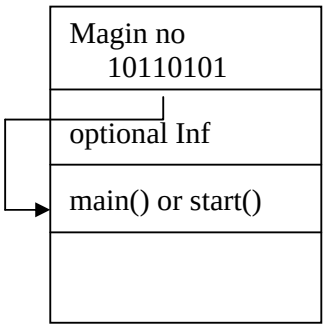
JDK supports on windows with Intel(platform) m/c's (o/s's).
JDK supports on Sun Solaris on spare m/c's.
JDK supports on Sun Solaris on Intel m/c's.
JDK supports on Linux on Intel m/c's.

Third-party vendors who are developing the JDK ,JVM, and JRE:
Blackdown.org -- ported the JDK to support Linux.
Transvirtual -- Java on Linux m/c's.
Towerj -- Much more efficient than Sun Microsystems JDK and JRE.
JRocket -- Is the best for windows boxes.

BEOS -- An OS Like windows o/s.

Can i write the C program without main() function?
Yes, you can. In Common Object File (COF) format the magic number exists. In this some number format sends the cursor to the starting of your program. This can be checked and identified by the o/s according to the format.

We can use Start() or any other name instead of main().



Winmain() is used in windows o/s instead of main(). Windows can take more arguments than main() in 'C' Language which can take two arguments.

```
Class Hello{
Public static void main(String ar[]){
System.out.println("Hello World");
}
}// saved in file Hello.java
```

Compiler produces class Hello.class. Class name is same for source code, developed after the compilation.

When we execute java Hello JVM loads the Hello class file and starts executing the main method, and prints the output.

jview is the virtual machine developed by Microsoft.

javap is Java profiler, used in order to check the internals of the java file.

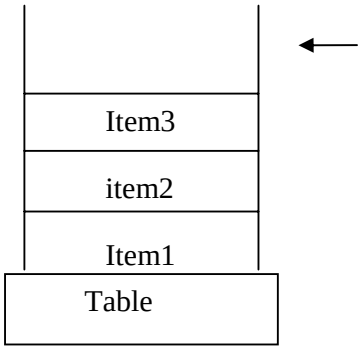
We can execute java -c <class file> , to observe the byte code

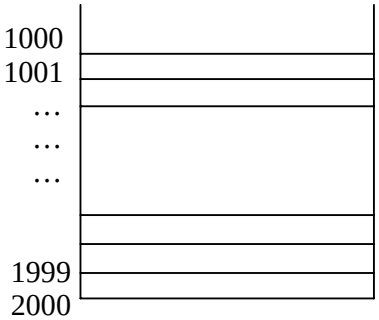
Actually Java is 8 bits (1 Byte) -- Internal virtual m/c's Storage is 32 bits that's why byte datatype can be stored as integer type.

Ex:	int	iadd
	float	fadd
	double	ldadd
	(32 bits)	(64 bits)

How exactly JVM performs arithmetic operations?

Stack based operation -> Last in first out.
Placing an item on a stack is known as pushing.
Removing an item from the stack is known as pop.
Specific memory area is known as stack (variables can be stored in this memory area).





push 100 -> 100 placed -> stack - 2000
push 200 -> 200 placed -> - 1999
push 300 -> 300 placed -> stack - 1998
pop 1 -> 300 removed -> stack - 1998
pop 2 -> 200 removed -> stack – 1999
pop 3 -> 100 removed -> stack – 2000

Byte codes are generated by the stack.
Stack is purely logical. Byte codes are generated by the stack.
According to the Java JVM performs all the operations on the stack.

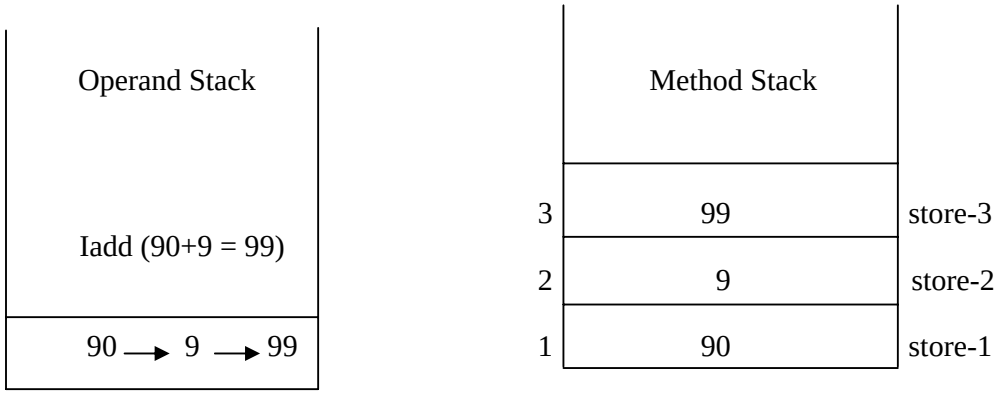
Operand Stack: is the place on stack where the JVM performs the operations.
Method Stack: is an area where JVM stores the local variables.

Java program:

```
Class Hello{
public static void main(String a[]{
System.out.println();
int i,j,k;
i = 90;
j = 9;
k = i+j;
System.out.println("k");
}
}
```

ByteCode:

```
method void main (java.lang.String[])
0 bi push 90   |
2 bi store_1   |---> constant for every int
3 bipush        9  |
5 istore_2     |
6 iload_1
7 iload_2
8 iadd
10 istore_3
11 return
```



int I value

Note:- push 90 -> places int i value 90 on operand stack.
store-1 -> stores the local variables on method stack , by removing the 90 from the operand stack.
Push 9 -> places int i value 9 on operand stack.
store_2 -> stores the variable 9 on method stack, by removing the 9 from the operand stack.
load_1 -> Both loads the variable values onto the operand stack, load_2 from method stack.

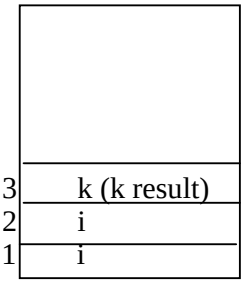
i add -> performs the addition operation on operand stack with the loaded values.
store_3 -> stores the result value after the operation on the operand stack, and stores it on the method stack.
return -> Take the result value from the method stack.
Load & add -> pop two data items from the method stack, add them and store the data again on the method stack.

Byte code can be created i many ways. There is no specific rule to create a byte code because it is created according the program which we have written.

k=i+j+k; :-Java compiler automatically creates the temporary variable and stores the result in it.

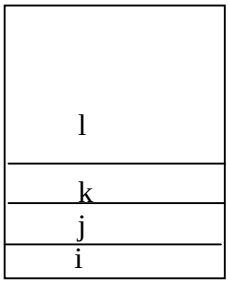
1) k=i+j+k;

```
iload_1
iload_2
iadd
iload_3
iadd
istore_3
return
```



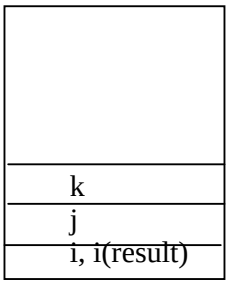
2)l=i+j+k;

```
iload_1
iload_2
iadd
iload_3
iadd
istore_4
return
```



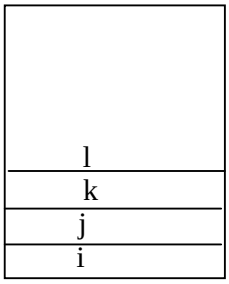
3)i=i+j+k;

```
istore_3
iload_1
iload_2
imul
iload_3
iadd
istore_1
return
```



4) i=(i+j)*k;

```
istore_3
iload_1
iload_2
iadd
iload_3
imul
istore_4
return
```



Can you write a program that run on JVM without using java, c, c++ languages?
Yes, first write the java class file (JCF) in an assembly language, and use the tool to convert this into java code which can be run on JVM.

Source (addition program) code written and compiled in 'c' language, should it be 100% portable to all the other platforms?
No.
C language is designed to explore the underlying hardware. Some platforms works on 16-bit machines, which can take the int as 2 bytes, and some are 32-bit machines, which can take int as 4 bytes. In 32-bit machines the range for int value is greater, but it is out of range to the 16-bit machines.

Portable:- Something it is easy to carry out. But we can write 100% portable to all the platforms by declaring 'long int' in 16-bit machines and just 'int' in 32-bit machines.

While designing the programs that are 100% portable to all the platforms, the designer must be careful.Knowledge about the operating system and the underlying hardware.
16-bit m/c's --> The m/c's which performs 16-bit operations.
32-bit m/c's --> The m/c's which performs 32-bit operations.

Design concepts of Java:-

The length of an integer datatype in java is 32-bit(Java is not with loose specifications) i.e. 4 bytes.

The length of a character datatype in java is 16-bits i.e. 2 bytes.

Java uses Unicode character set instead of ASCII, which is universally accepted standard. Here every char requires 16-bits.

ASCII:-is 7bit + 1(0(last memory location)):-8 bits:-1 byte.

In Unicode character set,there are codes for Telugu alphabets also. (Refer www.unicode.org)

For char datatype
1 byte:-for some characters - frequently used.
2 byte:-for some other char.
3 byte:-for some other char - less frequently used.
3 bytes generally used for Chinese and Japanese languages,because they are very lengthy characters.
Irrespective of the underlying platform following are stored in mentioned bit size.
Float - 32 bits.
Long - 64 bits.
Double - 64 bits.

Primitive datatypes in java:-

1.Numeric datatype:
Floating point
float (32 bits) double (64 bits)
b) Integer
byte (8 bits) short (16 bits) integer (32 bits) char (16 bits) long (64 bits)

Max range of values stored by byte < short < int < long datatypes.

Char datatype is considered as unsigned Integer, because it takes only +ve values.

Int:- a)signed Int, b)unsigned Int.
(stores +,-) (stores only +ve values)

2.Boolean datatype: (TRUE or FALSE)

This will reduces the no of paths.

Generic equation for max value i.e.
Stored in bytes = 2^(n-1)-1.

Min value store = -2^(n-1).
Note:- This is not applicable to the char datatypes.

Which datatype int or float stores greater values?
a)The way in which floating point datatype stores the value is different than the way in which integer datatype stores the value.
Ex:-value is 900000
X900
sign
X905 (9x10^5)
Float datatypes stores much greater
Value than an int datatype.
Note:- In float datatype notation, we are loosing the precision value.
Ex:- 9.99x10^3:- float datatype stores the value as X903 in 4
bits.

Java represents floating point number in the form IEEE 754
EEE: Electronics & Electrical Engineers.

754:- precision value.

Int byte are declared at run-time, how many bits of memory taken by each of them?
According to Java Byte should occupy 8 bits, short-16 bit, int-32 bits. But there is no special instruction to operate datatypes. All these datatypes internally treated as int datatype.

A virtual m/c implementer may choose to represent a byte available internally using 32-bits.But, the JVM implementer has to choose that a value less than -128 or >127 can never be stored inside a variable that is declare as byte.
But there is a special support for arrays of type byte, char, short and int.
Ex:-

```
class Hello{
public static void main(String args[ ]){
Byte i,j,k;
i=90;
j=9;
k=i+j;
k=(byte)(i+j);
}
}
```

Equivalent Byte code:

```
0 bipush 90
2 bistore_1
3 bipush 9
5 bistore_2
6 biload_1
7 biload_2
8 i2b
9 iadd
10 istore_3
11 return
```

Error will occur, external type casting is required. Because internally i,j treats as an int ,so byte cannot store int values.
Note: There are no instructions from byte to in, but there are instructions from int to byte & int to float.

i2b in byte code --> int to byte conversion.

Conversion of values from one datatype to another datatype:
1. Bigtype 2. Smalltype

When we convert a big data type to small data type there is a chance that we may loose some information. So, almost all the languages enforce the developer to use a 'type cast' operator while converting big data type to small data type.
In Java variables cannot be used without initialization.

Ex:- float f=99.99f (small f or capital F).

Automatic initialization of a=variables takes more time.

But in 'C' language variables automatically get initialized because C is high performance language.

How to read the syntax specifications?
Literals:- Source code representation of datatypes in a language is known as "Literals".
Ex:-float f;
int i,j;
boolean b;
i=64;
f=22.2f;
b=TRUE;

There are 3 ways of representation an int-literals.

Decimal Integer literals (0 to 9) - DIL
Octal integer literals (0 to 7) - OIL
Hexadecimal integer literals (0 to 9 & A to F) - HIL

This is one type of representing an int-literals, and the another type is:
DIL|OIL|HIL

Decimal Int-Literals:
decimal digit
decimal digit decimal digits
(or)
decimal digit(decimal digits)*
--> 0 or more decimal digits.
decimal digit (decimal digits)?
? --> 1 or more decimal digits.

Parsing: Reading the code and printing it in the meaningful units is known as ""parsing .

Ex:- int i = 64 ;
1 2 3 4 5 (all these are known as 'tokens").

2. Octal Int-Literals:- O is octal digit (0 is mandatory)
Ex:- int i= 0723;
octal digit
octal digit octal digits
(or)
octal digit (octal digits)* --> 0 or more
octal digit (octal digits)? --> 1 or more

Hexadecimal Int-Literals:-
OX
(or)
0[x|x](hexadecimal digits)*
0[x|x](hexadecimal digits)?

Construct the compiler is known as "compiler compilers".
'Lexus' is the tool that is used to construct the compiler.

Character Literals:-
Char ch= 'character';
|-->(A to Z | a to z).

float & double:-The way in which you write the double literals is same as the floating point literals. Because of this you should use
float f= 22.2f or F to represent the floating point literals.

Ex:- 1) double d=22.2f
l.d.t. = s.d.t. --> safe.

2) int i= 22.2
s.d.t = l.d.t. --> unsafe.

if(i<j) //if i & j are integers
{
i=10; j=20;
}
if(10<20) --> returns 1 --> true(other than zero).
If(20<10) --> returns 0 --> false

Any expression that used integer result in the test condition is known as "Integer Expression". This is for 'C' - language but in Java Boolean expression will be used inside the if statement.

if(Boolean Expression){
...
...
}

if(i=j) --> false
if(true) --> valid in java

if (22) --> not valid in java, but valid in 'C'.

There are two things that a compiler generally does:
The compiler accepts:- The language designer, while designing the language specifies the language conversion from one datatype to another.

The compiler may rejects: the language conversion cannot be done in this case.

There are two boolean variables b1 and b2. Can I add these two variables (b1+b2 or b1>b2 or b1<b2)?
We cannot add these two boolean variables. '<' or '>' should not be allowed with boolean datatypes. There is no such type of conversion.
But b1==b2 is acceptable.

Composition of datatypes:

When language designers design a language they define a set of primitive datatypes, for every primitive datatype they define composition of datatypes and the operations that can be operated on a specific datatypes.

If statement:-
s>if(test condition)
statement | block of statements
(no:of statements enclosed within the braces({ })).
{
stmt1;
stmt2;
|
|
|
stmtn;
}

If-else :-
if condition
stmt;
else
statement | block of statements;

Note: There is no 'If-elseif' statement in any language. A language designer need not define 'if-elseif' statement when a programmer writes 'if-elseif' statements the compiler actually uses the definitions of if-else and if statements internally.

Switch:- s> *Switch(Integer expression)*
{
case integerconstant:
|--> case label
statement; }
A Switch statement can take an 'Integer Expression' and as a part of the case label we have to use an 'integer constant'.
In case of the switch statement First the 'integer expression' is evaluated and them the

result of the expression is compared with the label. If one of the case Label matches, the Statements following that case label will be executed till a break is encountered. If none of the case Label matches, then the statements following the default: will be executed.

Ex:- *int i=3;*
switch(i)
{
case 1:
System.out.println("one");
.....
case3:
System.out.println("Two");
default:
System.out.println("End of the Switch case ");
}

Shall the above program runs properly or there is any error?
Any Statement can be proceeded by a Label that's why in the above example, it accepts the case3: as a normal Label, but not as a case Label.

If a programmer writes case immediately followed by 3 instead of blank between case and 3, it treats as a normal Label and compiles the Statements. In such cases case3: can not be treated as case label .So the program executes and prints the statements followed by the default: case.
Ex:- *int j=9;*
switch(i)
{
case j:
System.out.println("Nine");
default:
System.out.println("Default case");
}

Here error will occur since constant Expression Required.
We can not use a variable whose value can be changed as a part of a case Label.
Ex:-
int j=9;
switch(i)
{
case 'j':
System.out.println("Nine");
case (int)22.2f:
System.out.println("float to integer"):
}
Here,'j' --> char constant internally converts into int constants. In the same way type casting is used to convert float value as int value. Both of them are correct.
In Java language, the final keyword is used to declare the constants.(once the value is assigned, that cannot be modified)

Ex: *final int j=9;*
...

...
case 'j':
(is correct)
Ex: *final int j;*
...
...
case j:
(is wrong)
final --> indicates that the value of a variable cannot be modified after assigning it to a value.
A 'final' variable can be used as a part of case label, but we have to make sure that 'final' variable is initialized before the case label.

Note: Switch only takes int constants as case labels, it can't take char constants. Compiler automatically converts char to int, that's why switch is accepting the char constants as the case labels.

Statement Expression List(SEL): Treated as for initialize or for update. SEL | SEL,statement expression.

SE: Definition
Assignment
preIncrementExpression
preDecrementExpression
postIncrementExpression
postDecrementExpression
MethodInvocation --> calling method on function class Instance creation Expression.

In for loop you can't write more that one declaration statement in the for initialization.
Ex:
for (int k=0,float f=1.12f;k<5;k++)
in the above float f=1.12f is not valid.

Whereas *for(int k=0,f=1;k<5;k++)* is valid

As part of forUpdate any number of for update statement can be written separated by commas.

As part of statement expression can be allowed in for init and for update.(Method invocation is also valid in for update).
Ex:- *for(i=0,i<5;i++,System.out.println("two"))*

In the case of for init and for update the execution occurs from left most expression to right most expression.

As part of the for update, we can use any thing that can be treated as statement expression. According to Java language syntax an "if statement" cannot be considered as statement expression.
Ex:
for(i=0;i<5;i++,if(i==3) System.out.println(i))
(In the above code if statement is invalid)

Comments in coding should be an average of 100% to 150% for every 10 lines of code. Try to write the comments before you start writing the function i.e. outside the function.

i=0;
j=i+k; these are known as self documented code.

We can also design the language without using 'for' and 'while' loops (using 'goto' label)

Ex:-
```
int i=0
loop: System.out.println("raj");
i++;
if(i<10)
goto loop;
```

A language can be designed by supporting a *'goto'* statement without using any looping construct like 'for and 'while' or 'do while'. Java does not support *'goto'* statement, *goto* is a keyword in Java.

While is a very good instead of for...loop when
for(; i<5;){
|------> for updation is according to the statements executed.
//statements
//statements
}
Almost any logic that requires 'for loop' can be implemented by using a 'while loop' also. In most of the cases when we can initialization and update statements,it is always advisable to use 'for loop' instead of 'while loop'.
Ex:
for( ){
for(--------){
goto loopout:

}
}

loopout:
'goto is used here to come out of both the loops(outer loop).If break; is used it comes out of the current loop. But the outer loop will get executed.

Example:-
```
for(i=1;i<5;i++)
{
System.out.println(i);
if(i==3)
break; continue;
System.out.println("raj");
}
```

```
for(i=1;i<5;i++){
System.out.println(i);
if(i!=3)
System.out.println("raj");  }
```
When a break; statement is used in a for loop, the execution of the loop is stopped when the break is executed.
When a *continue* statement is used inside a loop the statements following the *continue* will be skipped for that iterations.

When a break; statement is used without a label the loop i.e. initialized by the label will be broken.
Ex:-
```
xyz:     for(i=0;i<5;i++)
{
System.out.println(i);
for(j=0;j<5;j++)
{
System.out.println(j);
break xyz;
System.out.println("xyz");
}
System.out.println("ABC");
}
```

Note: break; should be used inside the loop, otherwise it will gives an error at runtime.
When a break is used with a label the compiler searches for the label in the same loop or in the outer loops.

Check whether java language allows using multiple labels for the single statement.
Try with this Ex: ABC:xyz:System.out.println("raj");
If there is no error, then java language allows

Labeled continue: Java language also supports labeled continues.

Ex:
```
xyz: for(i=1;i<5;i++){
System.out.println("one");
for(j=0;j<5;j++){
System.out.println("two");
if(j==2)
continue xyz;
System.out.println("three");
}
System.out.println("four");
}
```
Compile the above program with the condition j==2 and check the result. Also check the program with the condition (j==2)&&(i==2)

WHILE LOOP: We can write a program without using a for loop,i.e all our for... loop programs can be converted to while... loop.

Ex:- (Note: In A and B below the outputs will be same)

```
For(i=0;i<5;i++)
{
System.out.println(i);
}
```

```
int i;
i=0;
while(i<5)
{
System.out.println(i);
i=i+1;
}
```

```
Do...loop:
i=0;
while(i<5)
{
System.out.println(i);
i++;
}
```

```
i=0;
do
{
System.out.println(i);
i++;
while(i<5);
}
```

Find out the results of the two programs.

```
do
{
read int();
read int();
char c=getchar();
}while((c='y')||(c='y'));
```

```
while(true)
{
read int();
read int();
if(c!='y')
break;
}
```

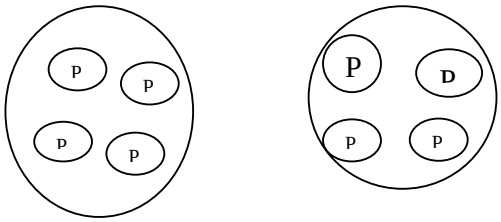
PROJECT DESIGN &DEVELOPMENT AND THE ERRORS OCCURED:

Come out with the designs such that a developer can easily identify the bugs which will reduce the total time taken to identify the bugs.

MODULAR DESIGN: Instead of designing the project or s/w as a single piece, design a product in n number of pieces. If any bugs occurred then that particular bugged piece can be modified and replaced. To do this it takes very less time when compared to the product as a single piece. Because of this total maintenance cost will be reduced.

The way in which one can see the design is different.

Developer1: Says that this design is a modular design.

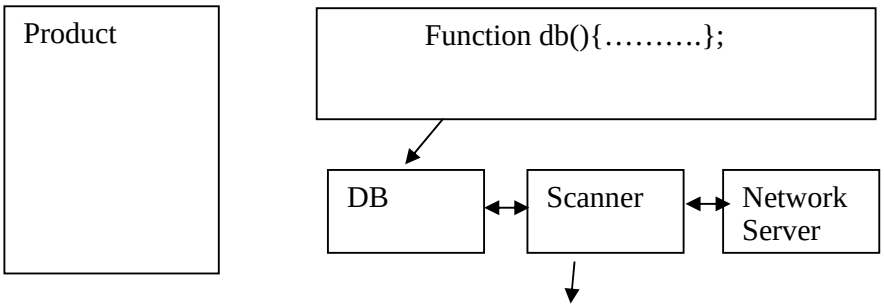


Developer2: Saw the design and find out that p2 and p4 depends on p1.So if any modification is done to the p1 should affect to both p2 and p14.In this case p1,p2 and p4 are treated as one part, and p3 treated as another part.

There should not be an interdependency in the parts designed because finding out the inter dependency in a software product is difficult.

Change of the code in one part or any part of the product requires the change of the code in another part is known as "Monolithic Design".

Write the code to improve the performance of the module in the module itself, instead of writing the ones developers) own functions.



If the product is developed with 3 modules, any interaction to that particular module should be done by writing the code in that particular module. In such a case any bugs that are occurred can be easily identified and that particular module can be tested and modified. Because of this time is reduced to identify the bugs.

Instead of writing like this, if a developer writes the separate function to interact with

the database, and the product is released to the client. After some time, client reports that he was getting an error while interacting with the database, and if one who developed that database module is not available. Then it is very difficult to the other person or a developer to identify that bug because he will never concentrate on the separate functions developed by the first developer.

Try to write the code to improve the performance according to the standard design given by the project manager, instead of writing your own functions.

Reasons why 'C' projects fails:(and why new languages are developed ?)

Most of the c projects fail because of the developers fault but not because of the language.

1)
```
Struct abc{
 int a;
int b;
int c;
 };
```
Composition: Set of variables declared in a structure. (what can be stored in the datatype)

2)
```
Struct Emp{
 int sno;
int basicsal;
int hra;
int da;
 };

hike_sal(structure Emp e, int hike){
e.basicsal=(e.basicsal+0.6*hra+0.4*da*hike)
};
```

Operations: hike(), changedept(), changeddesignation(), termination(), suspendTemporarily(), GrantLeave(), GrantLoan() etc.(actions that can be performed on the datatype)

In C language we can define our own datatypes by using structures, when a structure is defined. 'C' language developer can define the composition of the datatypes but not the operations that can be performed on that datatype.

For example an Emp datatype can be define by the developers, where the developer can decide how to store the information about an Emp in a program. But the programmer has no way of directly representing a set of operations that can be performed on Emp.

Developer1: Takes the document designed by the project manager.

Developer2: Develops the hikesal() function. It is working well. But if there are 10,000 records then this function is very slow. Then he use his own way to improve

the performance of the function hikesal(). He writes another function to improve the performance, or uses another datatype.

Developer2 has done this without consulting developer1 or Project Manager. If there are 10,000 functions it is very difficult to identify the bugs.
He performed the operations that are illegal, i.e. not according to the design document.

If the compiler restricts the user not to perform the user operations, then this type of mistakes should not be occurred.

When we are developing a project in 'C' language, we must design a project such that the document should be clear and indicating the composition of the datatypes and the operations that can be performed on that datatypes. For Ex. in a project the designer can clearly mention that hikesal(), calculatesal(), and changedept() should be called whenever an operation has to be performed on Emp datatype. if all the developers in a project follows the same rules, if at all a bug is reported related to Emp datatype, we can rectify the bug very easily. Since the operations an Emp performed only using the set of functions identified during the design.
Since 'C' language compiler doesn't restrict the developer to add his own code to perform some operations directly on a structure. Most of the developers using 'C' kind of language tend to make mistakes writing the code which directly access structures in their own way.

C++: Let us assume that there exists a new Language called 'NewC' language, which allows the developer to create his own datatype by defining composition and the operations that can be used on that datatype.

```
Struct Emp{
int eno;
int basicsal;
int hra;      //this defines the composition.
int da;
hikesal(){...}     //defines the operations.
calcsal();
};
```

In the above example the compiler knows that the legal operations that can be performed on Emp datatype.
e.hikesal() --> To call the operations.

With this setup developer2 cannot write his own code which operates directly on Emp datatype. If he finds that the hikesal() function running very slow, instead of writing his own code to increase the salary. He will suggest to do the modifications to hikesal() developed by developer1.

Note: Even by using C kind of language we can develop the applications properly, but there are very few restrictions. When a normal developer develops a project in C language instead of carefully using these restrictions they are misusing, ultimately resulting in so many bugs in the project.

Abstraction:

Ex: In Telephone Dialing a number is known as Interface and Complex circuit which takes the keystrokes of a user and perform the operation, is known as Implementation.

Any complex device can be operated by knowing the Interface, even if you don't know the Implementation.
In C language the prototype or definition (or the parameters or arguments that can be passed) of that particular function is known as "Interface".
The code written by the developer, what should be done when that function is called by the user is known as "Implementation".

Abstracting: Giving the details which are required to the user and hiding the complex details which are not required to the user. This is also allowed in Assembly Language.

Ex: Add, Multiply, Subtract, Division etc.

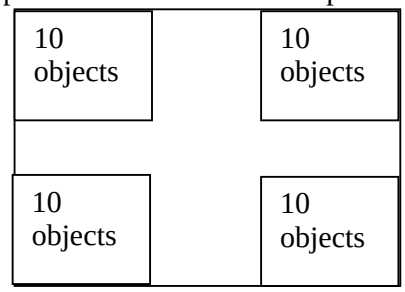
In a 'NewC' language ADT(Abstract Data Type) is used to define user datatypes like structure in "c".

```
ADT DATE {
int year;
int month;
int day;
sub_Date(){--}
sub_Days(){--}
add_Days(){--}
};
```

Encapsulation: Combining the compositions and operations together is known as "Encapsulation".
The process of hiding the complex details in a capsule is "Abstraction".

Ex: A chemical engineer combines the chemical composition and places it in a capsule to prepare a tablet.

Classification = Arrange in classes(or) categories. assign to class or category.
Generalization= Gives a general character or usually
Object = A material or thing that can seen or touched.
Class = Set of people or thing grouped together or differentiated form others.
Abstract = Not concrete; Existing in theory rather the practice.
Encapsulation = Enclose in a capsule or summarize.



Total there are 40 items or objects. But all these objects are classified into 4(types)

different classes.

Object:An entity of type to that particular class or an instance of a class.
Class: Blue print to create an object.

While designing a product: first
1. Come out with a design document--> class for the blueprint.
   Blueprint to create an object.
2. According to the class, we are designing the object.
3. Assemble all the products and create the final product or object.

When designing a mechanical instrument, we need to come out with the blue-print for the creation of different objects. When we apply the similar kind of design for software development we can express these blue-prints programmatically by using the abstract datatypes (ADT).

For Ex: To create a Mechanical Instrument cylinder, we need:
(Let us assume that these objects are required)
1. piston-one
2. crank_shaft-one
3. Connectin rod-one
4. piston rings-three
5. valves-two

To make a cylinder, we require 5 classes blue-prints to create 8 different objects.
Now let us assume that we are creating a software project called cylinder, we need to create 5 ADT' or classes.
Ex:
1) *ADT Piston{*
   float length;
   float radius; //properties as variables in software
   int materialtype;
   float thickness;

   moveup(){...} *//operations as functions*
   movedo wn(){...}
    }

2) *ADT Connectingrod{*
   float length;
   float diameter;
   float realradius;

   rotate(){--}
   }