JAVA_STUDY

I) Java General

1) Originally called 'Oak', which was to be used for embedded programs in consumer electronics
Java 1.0 was introduced in 1995, 1.1 in 1997 , 1.2 in 1998

2) Source Programs

- Are stored in Unicode characters(2 bytes), ASCII characters are 7 bits, Extended ASC is 8 bits
- The compiled java code is called byte code. When you run a java program, the byte co is sent to a Java Virtual Machine(JVM) which interprets the code and runs it.
- If the top-level class is public, the only way you get your source compiled is namin the source as *classname.java.* This is the common way of storing.
- If the top-level class is *friendly*, your source no need to be stored in *classname.java.*

4) **Encapsulation**

- Setting an accessible boundary and ability to hide data elements and methods of a class. Wayof hiding an data element by declaring, public, private, protected access modifiers
- Following factors are good candidates for encapsulation
  access to variables through methods
  no direct access to variables
  methods are public

5) **Polymorphism**
Ability to take many forms, applies only to methods and not to data members

6) **Inheritance**

- Ability to derive properties from base class (**private** elements from base class are no available to the derived class)
- Relationship between derived class and base class is **is a** relation ship
- Private members, constructors are not inherited.

7) Data Abstraction
Act of deciding what defines an object for the purposes of your program

8) **Class**
Blue print of object. Collection of variables and methods.

9) **Object** Instance of a class is object.

10) **Package names in java 1.2**

java.applet	java.rmi	javax.swing.filechooser
java.awt	java.rmi.activation	javax.swing.plaf
java.awt.color	java.rmi.dgc	javax.swing.plaf.basic
java.awt.datatransfer	java.rmi.registry	javax.swing.plaf.metal
java.awt.dnd	java.rmi.server	javax.swing.plaf.multi
java.awt.event	java.security	javax.swing.table
java.awt.font	java.security.acl	javax.swing.text
java.awt.geom	java.security.cert	javax.swing.text.html
java.awt.im	java.security.interfaces	javax.swing.text.html.parser
java.awt.image	java.security.spec	javax.swing.text.rtf
java.awt.image. renderable	java.sql	javax.swing.tree
java.awt.print	java.text	javax.swing.undo
java.beans	java.util	org.omg.CORBA
java.beans. beancontext	java.util.jar	org.omg.CORBA.DynAnyPackage
java.io	java.util.zip	org.omg.CORBA.ORBPackage
java.lang	java.accessibility	org.omg.CORBA.portable
java.lang.ref	javax.swing	org.omg.CORBA.TypeCodePackage
java.lang.reflect	javax.swing.border	org.omg.CosNaming
java.math	javax.swing.colorchooser	org.omg.CosNaming.NamingContext Package
java.net	javax.swing.event	

II Basics

1) Datatypes are

Data-typ	Bits	Declaration	Default	Miscellaneous	Divide by zero
boolean	8	boolean f=true;	false	can not be casted, boolean == boolean, is possible new Boolean(2) = false	not possible
byte	8	byte b1=0;	0	byte + byte gives int	RuntimeException
char	16	char c1='X'	'\u0000'	char++ is possible, String = char + String is possible char = char + String is not possib	
short	16	short s1=10;	0	short + short give int	RuntimeException
int	32	int i=10;	0		RuntimeException
float	32	float f1=100.0**F;**	0.0	float+int give float, **-0.0** in math f	Infinity
long	64	long l1=1000**L;**	0	above+long give long except float	RuntimeException
double	64	double d1=10.0;	0.0	above+double give double, **-0.0** in ma	Infinity
String		String s1;	null	String is an object	

Only instance(member) variables. Local variables and Objects must be initialized separately.
Array elements are initialized by default in both instance and Local variables.
Wrapper classes are *final, so those can not be extended.*

2) Keywords of Java(true and false are not keywords, they are boolean literals, null is nu
literal)

abstract	const(not currently used)	finally	int	public	this
boolean	continue	float	interface	return	throw
break	**default**	for	long	short	throws
byte	do	goto(not currently used)	native	static	transient
case	double	if	new	**strictfp (added for java 2)**	try
catch	else	**implements**	package	super	void
char	extends	import	private	switch	volatile
class	final	instanceof	protected	**synchronized**	while

3) Access Modifiers are public, protected, *friendly,* private

If a feature does not have any modifier, it is accessible only within a package, *this is friendl , but there is*
*no such keyword as friendly. (*Remember Mnemonic **NSSF)**

Other-modifiers	Applicable to
static	instance variables, methods, inner class, static initializer(instance blocks) not for interface and abstract methods
final	class, variables, methods , not for interface and abstract methods, not for vola
abstract	class and methods
volatile	variables, in multi processor environment modified asynch, not for interface , f
transient	variables (prevents serialization if present)
native	methods (to call outside of JVM like C/C++ function calling) loaded using System.loadLibrary("Libraryname"), not for interface and abstract methods native does not make a class to be abstract
synchronized	methods, and blocks (for threading), not for interface and abstract methods

4) Identifiers can start with $, _, Alphabetic characters
Length of identifiers can be any number
First letter can not be a number. It can be underscore or dollar sign or alpha

5) Operators precedence and hierarchy

Unary	postfix++, --, +, -,! ~, prefix ++,--	-(-5) is +
Creation, casting	new, ()	
Arthmetic	* / % + -	3 % 5 gives 3
Shift	<< >> >>>	3 << 1 is = 3 * 2
Comparison	< <= > >= instanceof == !=	String s1.equals(null) is invalid, but s1 == null is vali
Bitwise	& ^ |	
Short-Circuit	&& ||	
Ternary	?:	a=x?3:4;
Assignment	= "op="	Assignment associativity is right to left

6. Signature
- Method name + argument Nos + argument type + argument sequence is Signature
- throws is not a part of signature
- Return type is not a part of signature

7. java.lang.* classes will be imported by default.

Math - class is a public final class, it has public final static variables and public static methods

a. pow :- public static double pow**(double a, double b)**

8. Statement order is

```
package packagename;
import  java.util.*;
public class
```

9. A home **is a** house which **has a** *family and a pet*, Should be written as

```
public class home extends house{
   Family f;
   Pet    p;
}
```

10. ***instanceof*** is an operator. It returns true if the class of the

lhs operand is same as, or some subclass of , the class specified by the *rhs* operand.
rhs operand could be an interface. In such case, the test determines, if the object at the *lhs*
argument implements the *rhs* interface

if (x instanceof Component[]

Checks if object(x) is an array,and also determines the element type of x is some objec of Component

It is not valid to use like, if (x instanceof [])

For this we can use *OBJECT.getClass().isArray()*

11. XOR (^)

```
0 0  is    0
1 1  is    0
0 1  is    1
1 0  is    1
```

but ^= together if used as below, swaps

```
aa ^= bb;
bb ^= aa;
aa ^= bb;
```

12. InnerClass. Inner classes are also called as nested classes.

a. Innerclass can be public, private, protected, static, and abstract
   Inner inteface is allowed only inside a inner static class. There is no such thing as local, non-static, anonymous intefaces

b. Syntax

```
public class NonStaticTopOuter{            //top level class can not be static
  private int x;
  public  int i;
  public static int j;

  public class NonStaticInnerOne{
    private int y;
  //public static int y1;                  //static var is not allowed in non-static inner
    public void innerMethod(){
      x++;                                 //non-static variable of outer is okk
      j++;                               //static variable of outer is okk
      y++;                                 //variable of inner is ok
    }
  }

  public static class StaticInnerTwo{       //inner class can be static
    private int y;
    public static int y1;                  //static var is ok in static inner

    public void innerMethod(){
      //++;                              //non-static var of outer inside a static inner is not ok
      j++;                                 //static var of outer inside a static inner is okk
      y++;                                 //non-static inner var is ok inside static inner
    }
  }

  public void outerMethod(){
    System.out.println(" x is " + x);
  }
}                                        //end of outer class
```

In program above classes are referred as **NonStaticTopOuter & NonStaticTopOuter.NonStaticInnerOne**

```
public class DoOuterOneInnerOne{

public static void main(String[] args){
    NonStaticTopOuter o1=new NonStaticTopOuter();

//first way to define non-static inner class instance
    NonStaticTopOuter.NonStaticInnerOne i1=o1.new NonStaticInnerOne();

//second way to define non-static inner class instance
    NonStaticTopOuter.NonStaticInnerOne i2=new NonStaticTopOuter().new NonStaticInnerOne();

//way to define a STATIC INNER CLASS instance
//Static inner class does not have any reference to an enclosing class instance.
   NonStaticTopOuter.StaticInnerTwo i3=new NonStaticTopOuter.StaticInnerTwo();
  }
}
```

In directory above classes are stored as **NonStaticTopOuter.class**
NonStaticTopOuter$NonStaticInnerOne.class
NonStaticTopOuter$NonStaticInnerTwo.class

d. **Local Inner Classes :** These are classes defined inside a method. They can not use acce modifiers other than final. Enclosing method determines the class static, or non-static. Local class can use variables defined in enclosed class and the variables defined as ***final*** inside the method.

```
public void abc(int a;final int b){
  class Minner{
    public void method(){
        y=a+b       //is illegal, a cannot be used but b can be used as b is final
    }
  }     // Minner class end
}         //abc method end
```

e .**Anonymous Classes**

Declared and instantiated at same place

It takes the form

```
new Xxx(){ //body }          Where Xxx is an interface/class
new Xxx("asdfdsf") { //body}   where Xxx is an superclass
```

We can not write constructors, but compile will put the default constructor

Anonymous Arrays :

```
new int[] {3,4,5,6,7,8}      is valid
```

13. **STATIC**

a) Free-floating block will be run **only once** when class is loaded(static initializer)

```
static { // }         //can use only static variables inside, block can not have
```

any modifiers

Like this you can have instance initializer; like
{ //} for every instance that is created

b) static methods don't have **this, super**
c) static methods can use only static instance variables and not non-static variables. If it needs to be used, use it with object reference.
d) Static methods can not be overridden to be non-static and also the other way
e) top level classes can not be static. Inner classes can be static.
E.g.,

14. Abstract class can not be instantiated.

```
public abstract class Abstract1{              // abstract key word is must
  int i=1;
  public abstract void Abstract1Method1(int i); // abstract method can not have {}

  public void Abstract1Method2(int i){        // no need to over ride this
    System.out.println("Hello");
  }
}
```

For a class declared like above, you have to override Abstract1Method1(), and no need to override Abstract1Method2
Abstract method can not be final, static, synchronized, native
Abstract Class can have static, main method and can be run.

15. **Interface**

a. cannot be instantiated
b. Constants in interface are **public, static and final** by default
c. Methods in interface are **public and abstract and NOT *static*** by default, you can not change it
d. Interfaces can be extended from other interfaces using **extends**
e.g., public interface MyInterface extends FirstInterface, SecondInteface

e. all methods in an interface have to be implemented in implementing class
f. if we miss to some of the methods, we need to define the class as abstract
g. We can use an interface name to define a variable
MyInteface myOne[];
h. Interface methods **can't be**
native, static, synchronized, final, private, or protected
i. When interface methods are overridden, overriding methods can have
synchronized, native, final

16. **final**

a. final, makes a class not to be inherited
b. final makes a variable constant, general practice is to give in ALL CAPITALS any
c. static final variable should be initialized at the declaration statement itself.
e. non-static final instance variable can be declared blank**(blank final)** ,but only to certain extent.
An initialization statement must follow in all of the constructors within same class
f. final local variable can be declared blank**(blank final)** ,but it must be initialized immediately
within that method.
g. final is not allowed for **volatile** variables
h. final methods can not be overridden

17. Conversion, Java default of converting one data type to another is conversion

a. All conversion of primitives takes place at compile time.
b. final does not need casting as long as it's value is with in the limit

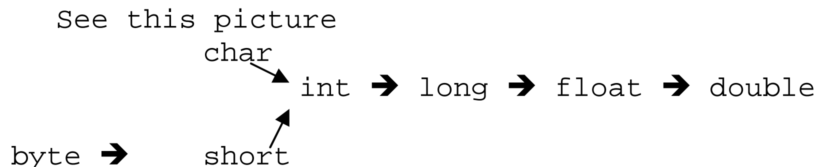
```
public  static  final int i=5;
public byte b=i;        // is ok, though i is int it's value is 5, with in byte limit
```

above will work with out the need for casting, because at the time of compilation itself, compiler knows the value and it can not be changed and it is with in byte limit.

c. There are 3 contexts in which conversion of primitive might occur
Assignment, Method Call, Arithmetic promotion

e. **<u>In Assignment</u>**, widening automatic conversion is possible
In Assignment, narrowing automatic conversion is not possible

See this picture



f. **<u>In Arithmetic, promotion</u>**

For op=, like +=, -=,   Conversion of *rhs* to *lhs* is automatic.
Interpretation way for x+=   should be as
x = (T) (x+y)   where T is type of *left hand side operand*

```
e.g.,  short s1=0;
       s1 + 10;          //will give int so explicit casting is needed, like
       (short) s1 + 10;
```

but   s1 += 10; will be converted to short automatically
also   s1++;   will be converted to short automatically

g. For unary operators byte, short, char ➔ int **(Definitely not)**

it is given like above in Simon Roberts then if,
byte b1=5;
byte b2=b1++;
should not work, but it works, which means we have to interpret like below
byte b2=(byte) (b1+1);

Any other type not converted

h. For binary operators,

If one of the operands is double, the other operand is converted into double
If one of the operands is float, the other operand is converted into float
If one of the operands is long,   the other operand is converted into a long

If anything else, both operands are converted to ints.

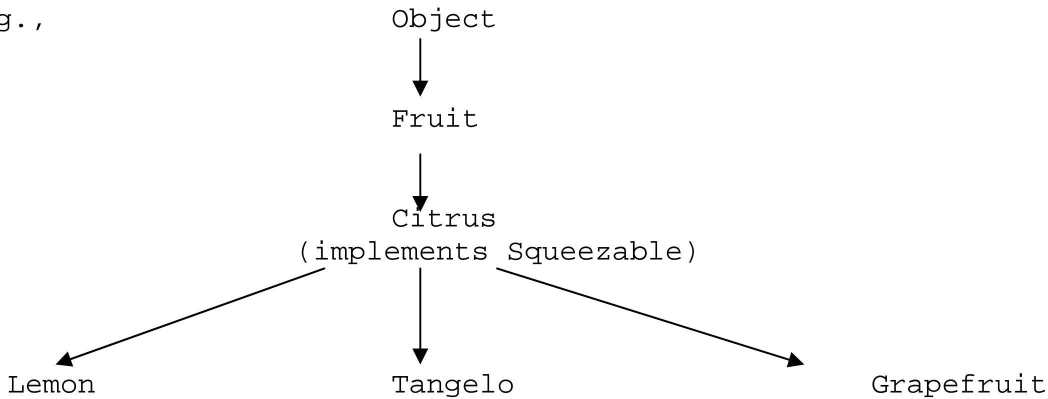
i. Object Conversion

3 types of object reference type; class type, interface type, array type

```
if RHS x = new RHS()
    LHS y= x;
```

	RHS is a class	**RHS is an Interface**	**RHS is an array**
LHS is a class	RHS must be a subclass of LHS	LHS must be **Object**	LHS must be **Objec**
LHS is an interface	RHS must implement interface LHS	RHS must a sub interface of LHS	LHS must be **Clone**
LHS is an array	Compile error	Compile error	RHS must be an ar some object refer type that can be converted to what LHSype is an arra

E.g.,



Consider the inheritance hierarchy shown in Figure above.

Code 1

```
Tangelo tange = new Tangelo();
Citrus cit = tange;                //This is fine because Citrus is parent of Tangelo
```

Code 2

```
Citrus cit = new Citrus();
Tangelo tange=cit      ;              //This is wrong,
```

Code 3

```
Grapefruit g = new Grapefruit();
Squeezable squee=g;              //This is fine
Grapefruit g2=squee              //This is wrong
```

Consider an example with arrays;

```
Fruits fruits[];
Lemon lemons[];
```

```
Citruses citrueses[] = new Citrus[10];

for (int i=0;i<10;i++)
     citruses[I] = new Citrus();

fruits = citruses                         //This is fine
lemons=citruses;                     //error
```

Object Casting

	RHS is a non-final clas	RHS is a final class	RHS is an interface	RHS is an array
LHS is a non-fin class	RHS must exten LHS or vicever	RHS must extend LHS	Always ok	RHS must be Object
LHS is a final class	LHS must extend RHS	RHS and LHS must be of same type	LHS must implement interface RHS	Compile error
LHS is a interfa	Always ok	RHS must implement LHS	Always ok	Compile error
LHS is a array	LHS must be Object	Compile error	Compile error	RHS must be an array of some type that can cast to whatever LHS is an array of

Illegal cast gives compiler error but, when objects are stored in a Vector and casted illegaly, RuntimeException : java.lang.ClassCastException: *Classname* is thrown

18.Overloading and Overriding

a. Overloading is in same class with different signature
   Overriding is related directly to subclassing and exactly same method declaration

b. Overloading is multiple implementation of the same essential functionality to use the same name
   Overriding, modifies the implementation of a particular piece of behavior of a subclass

c. Overriding can not be less accessible.   Order of accessibility is
   private ➔ friendly ➔ protected ➔ public

d. Static method can not be overridden with non static. Static method can be ***HIDDEN.*** ***for e.g.,***

```
public class ParentClass{
  public static void staticMethod(){
    System.out.println("Parent Class staticMethod");
  }
  public void nonStaticMethod(){
    System.out.println("Parent Class nonStaticMethod");
```

```
    }

  }

  public class ChildClass extends ParentClass{
    public static  void staticMethod() {
      System.out.println("Child Class staticMethod");
    }

    public void nonStaticMethod(){
      System.out.println("Child Class nonStaticMethod");
    }

  }


  public class DoChildClass{
    public static void main(String args[]){
      ParentClass p =new ParentClass();
      ParentClass p1=new ChildClass();
      ChildClass  p2=new ChildClass();
      p.staticMethod();
      p1.staticMethod();
      p2.staticMethod();

      p.nonstaticMethod();
      p1.nonstaticMethod();
      p2.nonstaticMethod();

    }
   }

  will give the output  as;

  ParentClass staticMethod
  ParentClass staticMethod
  ChildClass staticMethod

  ParentClass nonStaticMethod
  ChildClass nonStaticMethod
  ChildClass nonStaticMethod
```

because the invocation of **staticMethod**, when static uses at **compile time,** the **type of variable i.e., p1**, namely Parent, to figure out, which **static method** to invoke, whereas the invocation of nonStaticMethod uses the **class** of **p1**, namely Child, to figure out, at **run time**, which **instance method** to invoke.

19.Constructors are not inherited in subclasses

Constructor can be private, protected. private constructors can be used within that clas
only
Default Constructors will be provided only if other constructor is not present
super(), this() constructors if called should be the first statement inside a constructor();

If C extends B extends A and all three have method m(). You can invoke m() of B from
using super.m(),
but not the method m() on class A. Super() is **next up the tree**
super and this references are not available inside static methods.
Constructors can not be static, final, native, and abstract, synchronized
Constructors can not return anything, it can not have void also;

20. **String & StringBuffer**

a) Operations over String object will give new Strings(immutable)
Operation over StringBuffer object will change the current object(mutable)

String s1='xyz'; String s2='xyz'

(s1 == s2) returns true, string 'xyz' is placed in pool, and when s2 is executed, new
reference is created

But in below cases

String s1=new String('xyz'); String s2=new String('xyz');
(s1 == s2), returns false. and

StringBuffer s1=new StringBuffer('xyz'); StringBuffer s2=n
StringBuffer('xyz');
(s1 == s2), and s1.equals(s2) return false,
because StringBuffer doesn't have equals() method, so it used Object's equals meth
which
does compares address references.

s1.toString() == 'xyz' returns false too, so we have only one choice to compa
string buffers
s1.toString().equals('xyz'))

b) For String and String buffer, string position starts with 0,

substring(begin-index,end-index) ➔ end position is calculated as (end-index -1)

e.g., s="abcdefgh"
s.substring(3,5) gives output **de**

21. clone() is a protected method
If data members with in a class are class objects, those members are not cloned unless
you specify Cloneable, only references are copied
Steps to enable cloning
i) you have to implement java.lang.Cloneable interface in order to clone
ii) Override close() to make it public
iii) Call super.clone()

22. Escape characters are
\n - new line \r - carriage return \t - tab

\f - form feed \b - blank space

23. garbage collector can be called using System.gc()
 garbage collector runs object's finalize method before running gc
 gc() does not force garbage collection to run immediately

24. **main()** method
 a) main method can be called from another , provided the first main is public or friendly and called in same package
 b) static main method is allowed inside inner class
 c) static public void main(String args[])in outer class is also valid
 d) public static int main(String args[])
 public static int main(String args)
 will compile, but will give run time error java.lang.NoSuchMethodError: main

25. **Shift operators** >>> , << , >> work for short, byte int and long only
 for short, byte , int ,modulus of 32 is performed on right side operand
 for long,modulus of 64 is performed on right side operand

26.switch will take only byte, short, char, or int. float, double, objects, long are not allowed in switch. If byte, case value must be with in the limit of byte.
 switch can not use relational operators.

```
switch(x)             x can be expression
{
case 1                case can not be variable it has to be a constant
}
```

27.for(;;;) constructs a infinity loop

28.If you try to compile the following code, you will get compilation error

```
public class Test{
      public static void main(String args[]){
            char c='\u000a';           //invalid char constant
            System.out.println(c);
      }
}
```

it also gives error for '\u000d'.

Because Unicode escapes are processed very early, it is not correct to write '\u000a' for a character literal whose value is linefeed; the Unicode escape \u000a is transformed into an actual linefeed in translation step 1 and the linefeed becomes a LineTerminator in step 2, and so the character literal is not valid in step 3. Instead, one should use the escape sequence '\n'. Similarly, it is not correct to write '\u000d' for a character literal whose value is carriage return (CR). Instead, use '\r'. In Java, a character literal always represents exactly one character

In other words;
All java source files are interpreted by the compiler as Unicode. During compilation early on Unicode escape sequences are replaced by their Unicode characters. So the upshot is that by the time the compiler is interested in any character literals your code fragment has been converted to:

```
public class Test{
      public static void main(String args[]){
            char c='
            ';
            System.out.println(c);
      }
}
```

However, \n or \r can be used as escapes for the ASCII new line and carriage return characters
for example

```
public class Test{
      public static void main( String[] args ){
       System.out.println( (int)('\n') );
       System.out.println( (int)('\r') );
       }
 }
```

should print out

10
13

So the thing to realize really is that not all escape sequences get interpreted at the same time, the Unicode escapes get converted much earlier than the special character escapes (**\n \r \b \t** \\ **\' \"** etc).

29. Octal (012) and Hexadecimal value(0X11) can be given to short, byte, int, long, double
30. Java uses pass by value
31. **Class** object of a Class can be obtained by ;
    Class.forName() ; Object.getClass(); common.*clasname*.class

II Exception

a) Errors, Checked Exception, Runtime Exceptions
b) Errors are prblems that your are not expected to handle like
   ThreadDeath, LinkateError, VirtualMachineError
c) Runtime Exceptions need not to be handled
c) To create your own exception implement, **Exception** or **Throwable**

```
if (condition)
     throw (new userException())              // is the preferable way of
```

d) throws is the keyword used with a method to throw unhandled methods
e) Order is, If you have handled the thrown exception

```
try{}
catch{}
finally{}
statement next to finally{}. Program won't abend.
```

If you have not handled the thrown exception

```
try{}
finally{}
Program abends
```

f) If you have inner try blocks, the ones which are not thrown will be caught in

outer catch blocks, if mentioned.

g) throwing is not part of Signature
h) you can not catch an Exception that is never thrown inside try {}
compiler will not allow this
i) throws in method declaration and not really throwing inside a method is not a problem
j) throw any RuntimeExcpetion() and not using throws - is not a problem
k) return or System.exit() may be present in a try block only when no more statements are present after catch and finally
l) return may be present in catch, though blocks following after try catch finally
m) Constructors can have try catch finally,
Constructors can have *throws*
n) try if not throwing any checked exception can leave and have just **finally**

III Threads

a. Threads are registered using start(), which runs run(). run() is not coded explicitly

b. Threads can execute its own run() method (extends Thread)

```
Class CounterThread extends Thread{}
CounterThread ct = new CounterThread();
ct.start();
```

c) Threads can execute the run method of some other object(implements Runnable)

```
Class CounterThread implements Runnable{}
CounterThread ct=new CounterThread();
Thread t=new Thread(ct);
```

d) Runnable interface is preferable because, if your class is subclass of Thread it can not be
subclass of others
Note: Runnable has only one method run()
e) When a run() returns, the thread has finished its task, and is considered dead
f) Dead threads can not be restarted
g) **Daemon threads are** threads those die when the parent thread(thread that created the thread) dies
Can be created by , for Thread's subclasses
Inside the constructor , *CALL setDaemon(true)*
Can be created by, for Runnable interface classes
By calling, *object.setDaemon(true)*
Can be started only before thread starts, if started later, throws, IllegalThreadStateException

Threads those are created by a daemon thread will also be daemon threads

h) Thread states are

Running	- When thread is executing
Waiting	- Waiting, Sleeping., Suspended, Blocked
Ready	- not waiting for anything other than CPU
Dead	- all done

Any waiting thread will go to Ready state before going to Running state.

Except for synchronized code and the wait/notify sequence.

i) Default thread priority is **5**, range is from **1-10**.
j) Use Thread.currentThread to run methods like, sleep, getName() from Runnable implemented class
k) Thread.MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY
l) Thread control is art of moving threads from one state to other
m) For Applets, browser is main Thread
n) Pathways
Yielding -static yield() causes a Running thread to go to Ready state,and come back to the left state

Suspending & resuming - are now deprecated. suspend() suspends current thread, resume() resumes suspended thread

Sleeping & then waking up - static sleep()passes time without doing using CPU. Interrupt() will cause sleeping thread to go to Ready state

Blocking & then continuing - related to I/O, goes to waiting state

Waiting & then being notified
Monitor is any object that has some synchronized code.

Wait() puts an executing thread into the waiting state and release the synchronize locks
notify() and notifyAll() methods put waiting thread into the Ready state

o) Scheduling are two types, and is based on operating system
Preemptive - application has to pass from one thread to another by giving sleep()
Time-sliced or round robin scheduling - based on certain time threads will be run

p) Synchronization, Is of two types ,
Synchronized methods One synchronized method can be executed once at a time, for one particular
object If a method is synchronized and static, only one method will be executing at one point of time

Synchronized blocks synchronized(*object/this*){//block}

IV Java.util.*
a) Collection is an interface - subinterfaces are List, Set
b) Tree guarentees ascending order
d) Vector, Map, Tree won't not take primitive data type, take only Objects. Use Wrapper classes
d) int position=**indexOf**(object) searches a Vector for the presence of passed object, if -1 is
returned it means object is not found in the Vector

Type	Insert, Retrieve	Ordered, Unique	Description
Collections	nothing	no order,no unique (	
Set	interface	no order, unique (	
TreeSet	add, iterator	**ordered**	

BitSet	add, get		default value is false
Sequence	No keys	ordered, no unique (	
LinkedList	add, get, remove		
Vector	add, addElement,(	**added order**	*vector*.elements() returns Enumera *vector.iterator()* returns Iterato
Stack	push, pop		
Map	Searched by **Key**	ordered, unique (	
HashMap/HashTa	put, get	**not ordered**	HashMap allows null, HashTable do
TreeMap	put, get	**ordered**	

HashTable

a. In HashTable duplicate keys are stored if given, care should be taken while storing keys

b. h*ashmapvariable*.elements() returns Enumeration for a hashtable
   hashmapvariable.keys() return keys enumeration of a hashtable

c. Technique to return **keys** as an iterator
   Set s1=*hashmapvariable*.keySet();
   Iterator i1=*s1.iterator();*

   Technique to return **values** of an hashtable as an iterator
   Collection c1=hashmapvariable.values();
   iterator i2=c1.iterator();

V Date Manipulation

Define objects like,

```
Date()     date1 = new Date();                //belongs to java.util.*
DateFormat fmt1  = DateFormat.getDateInstance();   //belongs to    java.text.*
String     s1    = fmt1.format(date1);
```

Or define objects using either one of the below

```
Calendar            calendar = Calendar.getInstance() //returns GregorianCalendar
GregorianCalendar   calendar = new GregorianCalendar()
Date                date1=calendar.getTime();
DateFormatSymbols have methods like getMonths(),getWeeks() is in java.text.*;
```

VI AWT

Cursor belongs to java.awt and Border belongs to javax.swing.border.*

LAYOUT Managers

Layout Manager determines how the components are arranged in a container

a) FlowLayout – never changes the size of component, in otherwords, honors preferred size of the component.
   components will be the size of the label

   Panel's default Layout is FlowLayOut, thereby

Applet's default layout is FlowLayout

b) BorderLayout -
Frame's default Layout is BorderLayout
BorderLayout - Default for Jframe, JApplet, JDialog

BorderLayout.CENTER fills the unused area, this is default
BorderLayout is default for WINDOW, Frame

c) GridLayout, rows and columns
won't honor components preferred sized
d) CardLayout - One above the other, first added panel on top.
e) GridBagLayout -
GridBagLayout should be constructed as

```
GridBagLayout gridbag=new GridBagLayout();
GridBagConstraints constraint=new GridBagConstraints();
gridbag.setConstraints(component, constraint)
awindow.add(component)
```

fields of GridBagConstraints – clone is the only method

gridx, gridy, gridwidth, gridheight, fill, ipadx, ipady, insets, anchor, weightx, weight y

f) BoxLayout - not in Certification - javax.swing.*

Type	Constructor	Description **default**
Component	Abstract	a component can not be added to two con only the latest, gets displayed
1) Label	Label(String) Label(String, alignment)	CENTER, LEFT,RIGHT
2) Button	Button(String)	
3) Choice	just default constructor	
4) List	**List**(rows) **List**(int rows,boolean)	Adds a vertical ScrollBar if necessary rows is for visible items not total ite
5) TextComponent		
5.1)TextArea	TextArea(int rows, int cols) TextArea(String,rows,cols) TextArea(String,rows,Scrollbars)	SCROLLBARS_BOTH, SCROLLBARS_HORIZONTAL, SCROLLBARS_VERTICAL,SCROLLBARS_NONE
5.2)TextField	TextField(String,int cols) TextField(int cols),TextField(String)	
6) Scrollbar	Scrollbar(int orientation) Scrollbar(orientation,value,visible,min,max	VERTICAL, HORIZONTAL
7) Checkbox	Checkbox(String, boolean) Checkbox(String, CG, boolean) Checkbox(String, boolean, CG)	by default it is checkboxn when CheckboxGroup makes it as Radiobutton, if CheckboxGroup is null it is Checkbox
8)Canvas	**Canvas**(GraphicsConfiguration)	
CheckboxGroup	just default constructor, EXT OBJECT	CheckBoxGroup is not subclass of compon
Container	default constructor	container can contain other container instance of container cannot be created
1) Window	Window(Frame), Window(Window)	Window cannot be added to a container
1.1)Frame	Frame(String)	default layout is BorderLayout.

		Frame is top level window
1.2)Dialog	Dialog(Frame,String, boolean) Dialog(Dialog,String) Dialog(Dialog)	no default constructor
1.2.1)FileDia	FileDialog(String,mode) FileDialog(String)	LOAD, SAVE
2)Panel	Panel(LayoutManager)	default layout is FlowLayout
2.1)Applet	just default constructor	default layout is FlowLayout public void init() is the initalize met for applets. Applets can not have MenuB
3)ScrollPane	ScrollPane(int displaypolicy)	SCROLLBARS_AS_NEEEDED, SCROLLBARS_ALWAY SCROLLBARS_NEVER
MenuComponent	Abstract extends object	Menu can be added only to a Frame
1)MenuBar	just default constructor	MenuBar Creates the Bar
2)MenuItem	MenuItem(String) MenuItem(String, MenuShortCut)	Menu creates the Initial Drop downMenu MenuItem is to create individual menu i
2.1)Menu	Menu(String) Menu(String, boolean tearoff)	Panel cannot have menu and so Applet Menu.addSeperator() adds a separator li
2.1.1)PopupMe	PopupMenu(String)	Cannot be added to MenuBar. PopupMenu can be added to a Component
2.2)Checkbox MenuItem	CheckboxMenuItem(String, boolean) CheckboxMenuItem(String)	
Point	Point(int,int),Point(Point)	default creates at 0,0 postion
Polygon	**Polygon**(int[] xpoints, int[] ypoin, int npoints	
Rectangle	Rectangle(Dimension) Rectangle(int,width)	
Graphics	Abstract, just default constructor	has drawOval, drawPolyLine, drawLine

Events

a) Principle elements are
Event Classes, Event Objects and Event Listeners, Event Adapters

b) Handling Events is spreading events from event sources to event listeners
c) Providing appropriate actions in event listeners to deal with the events received
d) Super class of all events is java.util.EventObject
e) Event classes are in java.awt.event.*;
f) Events are based on **Event Delegation Model**

g) Steps involved in Event implementation on a component

1. implement the corresponding listener
2. create a listener object(*listenerobject)*
3. register the component by *component.addXXListener(listenerobject)*
4. implement the functionality of the corresponding method

h) Explicit event handling is done using enableEvents(AWTEvent.XXX_MASK)
Implement processXXXEvent(XXXEvent e) method;
super.processXXXEvent() should be called before returning in the corresponding processXXXEvent method

h) If addActionListener() adds a listener instance twice, that corresponding methods will run twice

```
addActionListener(listener1);
addActionListener(listener1);           // this won't give compilation and runtime erro

public void actionPerformed(ActionEvent e)    // will be performed twice
```

If enableEvents and corresponding listenere, both are included in a class(which is unnecessary
because when you addXXXlistener that will enableEvent that MASK also) both will run starting with enableEvents();

Images and Animation

1. Can be done in application and applets
2. Font(String fontname, int fontsize, int size)
   font name can be Serif, SansSerif, Monospaced
   font size can be Font.Plain, Font.Bold, Font.Italic
3. init()
   start()
   stop()
   destroy()
   update() - default update clears the applet, override with your update(Graphics g) calling paint()
   repaint() - calls default update and paint methods
   public void paint(Graphic g)

VII INPUT OUTPUT (java.io.*)

1. Java supports two types of streams, **binary** stream, **character** stream(Readers and Writers)
2. Binary stream writes data in bytes in binary form. This includes numeric as well as character data. Characters in binary stream are written as Unicode characters.
3. Character stream is read and written as text.
4. InputStream, OutputStream, Reader, Writer are all abstract classes
5. All of the above classes are serial in nature, they cannot process randomly
6. **File** represents physical file or directory on your hard drive and not a stream
7. Absolute path includes drive letter. Relative path omits drive letter and will be interpreted
   as a path relative to the current directory
8. to throw SecurityException(runtime) Security manager must exist on the local machine
9. getName() gives the name of the file and not path
10. getPath() gives path and file name
11. list() gives String[] of file names
12. listFile() gives File[] objects of directory and file names
13. list() accepts FilenameFilter object
14. listFiles() accepts FilenameFilter or FileFilter
15. Serialization was added in JDK 1.1 for distributed objects (RMI)
16. BufferedInputStream for better performance with slow sources

Semantic Events					
Event type	**Source**	**Listener Interfaces & Adapters**	**Listener Methods**	**Registration and removal**	**Explicit Event Handling enable(Events)**
ActionEvent	Button, List **MenuItem,** TextField	ActionListener	actionPerformed(ActionEvent)	addActionListener()	AWTEvent. ACTION_EVENT_MASK
AdjustmentEvent	ScrollBar	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)	addAdjustmentListener()	AWTEvent. ADJUSTMENT_EVENT_MASK
ItemEvent	Choice, check box, checkbox menu item, list	ItemListener	itemStateChanged(ItemEvent)	addItemListener()	AWTEvent. ITEM_EVENT_MASK
TextEvent	TextArea, TextField	TextListener	textValueChanged(TextEvent)	addTextListener()	AWTEvent. TEXT_EVENT_MASK
Low-Level Event					
Component Event	Component	ComponentListener ComponentAdapter	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)	addComponentListener()	AWTEvent COMPONENT_EVENT_MASK
ContainerEvent	Container	ContainerListener ContainerAdapter	componentAdded(ContainerEvent) componetRemoved(ContainerEvent)	addContainerListener()	AWTEvent CONTAINER_EVENT_MASK
FocusEvent	Component	FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)	addFocusListener	AWTEvent FOCUS_EVENT_MASK
KeyEvent	Component	KeyListener KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)	addKeyListener	AWTEvent .KEY_EVENT_MASK
MouseEvent	Component	MouseListener MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)	addMouseListener()	AWTEvent MOUSE_EVENT_MASK
	Component	MouseMotionListener MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)	addMouseMotionListener()	MOUSE_MOTION_EVENT_M ASK
WindowEvent	Component	WindowListener WindowAdapter	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent)	addWindowListener()	AWTEvent WINDOW_EVENT_MASK

			|windowOpened(WindowEvent)		

	Output	**Constructor**	**Exceptions Thrown**	**Miscellaneous**
Binary-OutputStream		abstract class		all methods return void
A)FileOutputStream	File		FileNotFoundException, r-SecurityException	getFD()
B)ByteArrayOutputStream	ByteArray		r-IllegalArgumentExceptio	toByteArray(), toString() default array = 32 bytes array automatically grows
C) PipedOutputStream	PipedInput		IOException	all methods return void
D) FilterOutputStream	OutputStream			
d1)DataOutputSream **has writeXXX**	above 3			size(), write() writes low order 8 bits.
d2)BufferedOutputStr	above 3			
d3)PrintStream	Console			checkError()
d4)ZippedOutputStrea	Zipped file			ZipEntry()
d5)CheckedOutputStre				getChecksum()
E) ObjectOutputStream has **writeXXX**	Serializable		IOException SecurityException	replaceObjects(), putFiel
Binary-InputStream -1 forend of stream		abstract		mark(),reset() are used to mark a position in the stream and return.
A)FileInputStream	file		c-FileNotFoundException r-SecurityException	
*B)*ByteArrayInputStream	array			
C)PipedInputStream	PipedOutput		IOException	
D)FilterInputStream				
d1)DataInputStream	from above 3			**has readXXX**
d2)BufferedInputStrea	from above 3			
d3) PushBackInputStre	from above 3		IllegalArgumentException	
d4) LineNumberInputSt		Deprecated		
d5) ZipInputStream	zipped input			has ZipEntry
E)ObjectInputStream	Serialized input	new ObjectInputStrea new FileInputStream(	StreamCorruptedException IOException, SecurityExce	**has readXXX**
F)SequenceInputStream	to concantenate input streams	**SequenceInputStream** (Enumeration) InputStream, InputStream2)		

	Output/Input is	**Constructor**	**Throws**	**Miscellaneous**

Character OutputStre		abstract		Conversion of character used by local computer **only write(), no writeXXX**
A)StringWriter				getBuffer(), toString()
B)CharArrayWriter	Char Array			toCharArray(), toString()
C)OutputStreamWriter	OutputStream			getEncoding()
c1) FileWriter	File			
D)BufferedWriter				must be closed
E)PrintWriter				checkError()
F)FilterWriter	very vast	protected	nothing	
G)PipedWriter	to PipedReader		nothing	checkError()
Character InputStrea		abstract		**only read()**,no readXXX()
A)StringReader	String			
B)CharArrayReader	Char Array		nothing	
C)InputStreamReader			UnsupportedEncodingExc	
c1) FileReader			FileNotFoundException	no methods
D)BufferedReader	Reader	BufferedReader(Reader,	IllegalArgumentExcepti	must be closed has readLine()
d1) LineNumberRead		LinenumberReader(Reade	nothing	
E)FilterReader		protected, abstract		
e1) PushBackReader		PushBackReader(Reader)	IllegalArgumentExcepti	has unread()
F)PipedReader	from PipedWriter	PipedReader(PipedWrite	IOException	connect(PipedWriterr) is key method
Formatted Stream				
StreamTokenizer	OutputStream	new StreamTokenizer(new InputStream**Reader**(System.in)		System.in, is preferred c reading characters from k board. **sval** *is for string is for numeric double*
RandomAccessFile	File	new RandomAccess(File, **"r"/"rw"**)		

VIII JDBC

a) For SQLs

i) Load driver Class.forName("driver name")
ii) Connection established DriverManager.getConnection(url,id,password)
iii) Prepare Statement using PrepareStatment stmt= connection.prepareStatement("Sql statement");
iv) Execute query ResultSet rset = stmt.executeQuery()

b) For Stored Procedures

i) Load driver Class.forName("driver name")
ii) Connection established DriverManager.getConnection(url,id,password)
iii) Callable Statement CallableStatement stmt=connection.prepareCall("CALL PROCEDURE_Name(?,?));
iv) Setinput parameter using setInt(parameter no, type)
v) Register OutputParameter using stmt.registerOutParameter(1,Types.INTEGER);
vi) execute using stmt.execute();

c) executeQuery() is used only for Select statements. executeUpdate(),executeInsert(), executeDelete() are used for corresponding operation which return the no of rows affected

d)

IX Servlets

Two packages
javax.servlet - GenericServlet
javax.servlet.http - HttpServlet

Methods which are important inside a Servlet program

public void **doGet**(HttpServletRequest req, HttpServletResponse res) throws ServletExcetpion, IOException
public void **doPost**(HttpServletRequest req, HttpServletResponse res) throws ServletExcetpion, IOException

req.getMethod() gives

Simple Servlet Example:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

SSI (Server-side include functionality) :- Preprocessing of a page by parsing .

```
<SERVLET CODE=servlet name CODEBASE=http://server:port/dir
        initparm1=value initparm2=value>

<PARAM NAME=param1 VALUE=value1>
<PARAM NAME=param2 VALUE=value2>
   if you see this text, it means, that the web servers, providing this page, does not support the
SERVLET TAG
</SERVLET>
```

Servlet receives the parameter value passed usign getParameter()

g) files must be stored with **.shtml** for JAVA WEB SERVER, for the parsing to take place

X JSP (JAVA SERVER PAGES

a) Allows direct insertion of servlet code in an HTML file.

b) Blocks (called scriplet) is surrounded by <% tag and a closing %> tag.

c) 4 pre defined variables

```
        request    - for HttpServletRequest object
        response   - HttpServletResponse object
        out        - PrintWriter object
        in         - BufferedReader
```

d) extension **must by .jsp**

e) Behind the scenes, the server compiles, loads, and runs a special servlet (**workhorse** servlet) to generate the page' content

Expressions and Directives

1) Expresssions begin with <%=, and ends with %>
2) Expressions are converted to a String
3) Directives begin gwith <@= , and ends with %>
4) Directives have following 6 variables to set
5) content_type ➔ eg., <%@ content_type ="text/plain" %>
   default is "text/plain"
   import ➔ eg., <%@ import ="java.io.*,java.util.Hashtable" %>
   extends ➔ eg., <@ extends="*superclassname*" %>
   implements ➔ eg., <@ implements="*interface name*" %>
   method ➔ eg., <@ method="doPost" %>
   language ➔ eg., <@ language="java" %>

Declarations

To define non local variables and methods in the workhorse servlet.

```
<SCRIPT RUNAT="server">  </SCRIPT>
```

In between you can include any servlet code that should be placed outside the service method.

e.g.,
```
<SCRIPT RUNAT="server">
        private static final String DEFAULT_NAME = "world"

        private String getName(HttpServerRequest req)
        {
              String name = req.getParameter("name");

              if (name ===null)
              {
              }
        }
</SCRIPT>
```

JavaBeans can be used in a JSP *page using* ***<BEAN>*** tag.

Single process Web Server has it JVAM embedded inside

Multi process web server does not really have the choice to embed a JVM directly in its process.

l)

XI Important web page addresses

a) Sun

java	-	www.java.sun.com
JDK 1.3 download (Windows)	-	http://java.sun.com/j2se/1.3/download-windows.html
JAVA API documentation (browse)	-	http://java.sun.com/j2se/1.3/docs/api/index.html
JAVA API documentation (download)	-	http://java.sun.com/j2se/1.3/docs.html

b) Mock Exam

Marcus green - www.jchq.net

c) Integrated Developing Environment (IDE)

Visual Café (Download-Free) - http://www.webgain.com/download/visualcafe/standard_edition_download.html

d) JavaSoft Website → www.javasoft.com

e) Java Developers Connection → http://developer.java.sun.com/developer