This document has material collected from a lot of resources. I have listed some of the resources here. I'm thankful to all those great people. Anything I forgot to mention is not intentional

1. e-book written by kathy siera and bert bates.
2. this study material of my own.
3. Marcus Green's website, http://www.jchq.net
4. javaranch discussion forums, http://www.javaranch.com
5. Java Language Specification from Sun - http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html
6. Java tutorial from Sun - http://java.sun.com/docs/books/tutorial/index.html
7. radhika palwai's website - http://www.geocities.com/r_palwai/scjp.htm
8. An ever increasing websites for SCJP exam.

Important resource:

One of the most important resources to me was www.yahoogroups.com, it contains several groups for java certification, I encourage subscribing in what you are interested in, but my advice is to browse the mails through the web not to receive individual or digested mails, because they are very active groups.

Some of the yahoo groups which I think are good are

jcertification-subscribe@yahoogroups.com
Only e-Group for preparation of certification for java programmers. Experts for answering the questions and discussions. Feel free to post your questions and get the best answers. Ask anything regarding Java Certification. Register today and tell a friend about this.

javacertstudy-subscribe@yahoogroups.com
JavaCert.com is an online study community for Sun's Java certification exams.

sunjavacert-subscribe@yahoogroups.com
The list basically provides tips for Java aspirants. A list with information on Java Certification, SCJP, SCJA , SCJD , tips , books , tutorials, downloads, lists, mock and online exams. A list that can give u that extra niche to get a high score in Java Certification.

If you found any error you just mail me at anand_bheemaraju@yahoo.com

Declarations and Access control :-

1. Following are invalid declarations,
   ```
   int a[3];
   int a[] = new int[];
   int a[][] = new int [][4];
   int a[] = new int[2]{1,2};
   ```

2. Remember these points,
   ```
   int [] a1 ,a2 ,a3;
   ```
   The above declaration symbolizes that a1, a2, a3 are dmensional arrays.

   Where as,
   ```
   int a1 [], a2 , a3 ;
   ```
   The above declaration symbolizes that a1 is a one dimensional array where as a2 and a3 are simple integer numbers.

   → General rule is that the big brackets i.e '[ ]' must appear after the type name like int , long etc or after the identifier name. If the '[]' appears after the type name then all the identifiers after that type name will be arrays of the specified type where as if the '[]' appears after the identifier name then it only qualifies as the array of the specified dimension.

   →array of a primitive type is a class derived from Object class itself and it also implements Cloneable and java.io.serializable.

   → '[]' cannot occur before the identifier name during declaration i.e
   ```
   int a1,[]a2,a3;//causes error because '[]' appears before a2 and not after a2.
   ```

3. While equating array reference variables if the dimensions aren't appropriate then compile time error is not caused but runtime error sets in.

4. In a given file only one public class can exist and also the name of the public class and the file must be the same.

5. protected , private access modifiers cannot be applied to top level classes.

6. among the other modifiers that cannot be applied to a top level class are static , synchronized ,native etc but we can apply abstract , final , strictfp.

7. An anonymous class is not local class i.e local class is always a named class. Furthur an anonymous class is always final.

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

8. A final field can be assigned a value only once (not necessary to do the initialization in the declaration part itself).

9. Compiler error is caused if the final instance variables which are declared aren't initialized by the end of each constructor.

10. if the final variable is method local , then even if it isn't initialized compiler won't complain ,unless it is used in some expression.

11. A regular inner class may be declared both abstract and private but not abstract and final.

12. Field variables and methods can share the same name as that of the enclosing class.

13. A nested class and the outer class can't share the same name.

14. But a nested class , method , field variable can share same name provided the names of the nested class and enclosing class are different.

15. A top level class can be declared abstract even if it doesn't contain any abstract methods. It can even extended one of concrete classes and can implement other interfaces.

16. Return types :-consider

```
byte m1(){
final int i = 1;
return i ;
}
```

here the method compiles fine as the integer variable is marked final and it's value is known at compile time.

```
byte m2(final int i){
return i ;
}
```

here though variable 'i' is marked final ,it's value is not determined at compile time and hence it results in compile time error.

Flow control, Assertions, and Exception Handling :-

1. To enable assertions on non – system classes use –ea or –enableassertions
2. To disable assertions on non – system classes use –da or –disableassertions
3. To enable assertions on system classes use –esa or –enablesystemassertions

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

4. To disable assertions on system classes use –dsa or –disablesystemassertions
5. Assertions can be selectively enabled for the unnamed package in the current working directory.
6. In any method with a non void return type we can't place an assert statement like "assert false;" ,it results in compiler error but we can place a "throw new AssertionError();" in place of a return statement.

7. A instance variable can be redeclared within a method ,like for example

```
class A{
int i=1;
void m(){int i=2;}
}
```

8. A method local variable cannot be redeclared within another block inside a method after declaring it inside the method ,like for example

```
class A{
void m(){
int i = 10;
for(int i = 1;i<20;i++){//this line causes compiler error
System.out.println(i);
}}}
```

9. If a method local variable can be declared inside a method after declaring it inside a block of code then no compiler sets in ,like for example

```
class A{
void m(){
for(int i=0;i<10;i++){
System.out.println(i);
}
int i=100;
}}
```

the above class when compiled doesn't cause error because the variable 'i' is declared inside the method after it is declared inside the for loop.

10. A constructor can have the following 4 access levels,
    - public
    - private
    - protected
    - default or package access level(no modifier)

    →If no constructor is supplied by the programmer then the compiler itself creates a "default constructor" for the class. The default has the following properties
    - it takes no argument
    - it contains a call to the no-arg constructor of the super class as its first statement.
    - it will have the same access level as that of the given class.

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

11. A constructor
    - cannot be declared final , static , native , synchronized.
    - It can declare any exception in it's throws clause.
    - if any other class extends the given class then that class must include the exception in its throws clause or must include a wider exception in its throws clause.
      → it cannot declare the call to super class constructor in try catch clause.

12. An anonymous class doesnot have any constructor.
13. initializer blocks cannot throw exceptions.

Garbage Collection

1. Garbage collection is a process which is entirely in the control of the JVM. The user can only suggest the JVM to carry out the garbage collection.
2. finalize method has the following prototype
   → protected void finalize()throws throwable

3. finalize method of the given object will called by the JVM only once before garbage collecting the given object. If the object is resurrected inside the finalize method then that object will not be garbage collected. But when the object becomes eligible to garbage collection later the finalize method of the object will not be called again.

4. if the finalize method throws any exception then JVM shall ignore it.

5. Garbage collection in java is vendor implemented.

Language Fundamentals :-

datatype	Default value
int	0
char	'\u0000'
float /double	0.0
Object reference	Null

Primitive datatype	Range
byte	-128 to 127
short	-32768 to 32767
int	-2147483648 to 2147483647
long	- 9223372036854775808 to 9223372036854775807
char	0 to 65535

1. following are valid declaration of package ,import and class declarations
   - package a;

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

import java.io.*;
class A{}

- import java.io.*;
  class A{}

2. following are invalid declarations ,
   - import …..
     package ….
     class ….

   - class…..
     import ….
     class….
3. the package or import statement at the beginning of the file is applicable to all the classes in the file.
4. following are valid main declarations
   - public static void main(String [] args){}
   - public static void main(String []XXX){}//in the place of XXX any thing can be placed
   - static public void main(String [] args){}
   - public static final void main(String [] args){}
   - public static synchronized void main(String [] args){}
   - public static strictfp void main(String [] args){}
5. following declarations of main method results in runtime error(not compile time error)
   - public void main(String [] args){}//static modifier missing
   - static void main(String [] args){}//public modifier missing
   - public static int main(String [] args){return 1;}//return type must be void
   - private static void main(String [] args){}//access modifier is private
   - protected static void main(String [] args){}//access modifier is protected
   - public static void main(String args){}//argument to main method is a String and not 'String[]' (String array)
   - public static void main(){}//argument String[] missing

6. Any class which implements a given interface must provide implementation to all the methods in the interface unless the class itself is abstract.

7. The Runnable interface has one and only one method whose prototype is

   - public void run();

8. Thread class implements Runnable interface.

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

9. When an array is created then the contents of the array are indexed from 0 to array.length-1 .

Java keywords

package	import	class	extends	interface
implements	public	private	protected	final
abstract	strictfp	super	this	byte
short	int	long	char	boolean
float	double	static	transient	volatile
void	return	native	synchronized	try
catch	finally	throw	throws	for
while	do	break	continue	switch
case	default	if	else	new
instanceof	const	goto	assert	

10. A variable declared within a method will not get it's default value and hence it if it is used in any expression then the compiler objects that the variable may not be initialized.
11. instance variables get their default value.
12. Array elements get default value whether it is declared inside a method or declared outside the method.
13. backslash characters are

'\b'	'\f'	'\n'	'\r'
'\t'	'\\'	'\''	'\"'
'\0'	'\1'	'\2'	'\3'
'\4'	'\5'	'\6'	'\7'

In any String or char if value present next to '\'(backslash) is other than those mentioned in the above table then it results in compiler error ,for eg
"\/santosh"//results in error
"/\santosh"//results in error
"//santosh"//no error
\\santosh //no error

14. given, \u000a = \n and \u000d = \r we cannot use these(\u000a and \u000d) in any string or char assignment statements. Hence the following will cause compiler error ,
String s1 = "\u000a";
String s2 = "\u000d";
These two are interpreted as line terminators and hence results in compiler error.

Overloading, Overriding, Runtime Type and Object Orientation

1. When a overloaded method is invoked then the reference type of the variable is considered and the decision as to which method has to be invoked is determined at compile time but if a overridden method is invoked then the run time type of

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

the object is considered and the decision as to which method has to be invoked is determined at runtime.

2. A tightly encapsulated class allows access to data only through accessor and mutator methods.
3. If a call to the super class constructor is made with argument i.e for example with the following prototype,
   super(ArgList);
   - Here the ArgList can be static variables or static methods belonging to the given class or one of its super class.
   - ArgList cannot refer to instance variables or instance methods belonging to the given class or to one of its super class.

4. The variables in the superclass can be hidden by variables in the given class.The variable that is accessed is determined at compile time based on the type of reference not based on the run-time type of the object.
5. A static method in a super class can be overridden by another static method in subclass. If a non-static method overrides a static method or if a static method overrides a non-static method then compile time error sets in.
6. If a method in a class is private then it cannot be overridden but can be redeclared inside the subclass extending the superclass. This is because the private method of the super class remains invisible to its subclasses.
7. When an object is being constructed then following two things occur
   - All the initializations of the static variables starting from supermost class(Object class) upto the given class occurs first.
   - This followed by the execution of the constructors starting from the supermost class (Object class)upto the given class occurs.

8. Just as <EnclosingClassName>.this.<VarName>; can access the variable of the enclosing class from inner class ,<EnclosingClassName>.super.<VarName> is used to access variable belonging to super class of the enclosing class.
9. When we have a number of classes forming a inheritance chain then the appropriate overridden method is determined by the run time type of the object where as if we consider static methods or fields then the appropriate one to be executed will be decide by the reference type.
10. If an inner class is non-static then static , non-static , private variables and methods of the enclosing class are accessible to the inner class even in the constructor.
11. static inner class can only access static variables and methods of enclosing class.
12. If a static – nested class i.e. top level nested class is instantiated using the form new <OuterClassName>.<InnerClassName>.();
    Then only the constructor of the inner class is executed.
13. Enclosing class members can access the private variables and methods of its member classes. These private fields and methods can be accessed by other member classes also.
14. A non-static Inner class cannot have static fields or methods , classes and interfaces (because nested interfaces are implicitly static).

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

15. The only static fields allowed in a non-static inner class are those marked final and whose value is determined at compile time.
    → here both declaration and initialization must happen in the same statement otherwise compiler error is caused.
16. If outer class i.e. enclosing class is alone instantiated then the constructor of its member classes are not executed.
17. 'this' and 'super' keywords cannot be used in a static method.
18. We can't instantiate a non-static inner class from a static method using the form, new <InnerClassName>();
    But through this way a static class can be instantiated in a static method or a non-static method.

Threads :-

1. Methods in object class
   - wait → throws interruptedException
   - notify
   - notifyAll

   Remember → they are final and must be called from within a synchronized context, otherwise illegalmonitorstate exception is thrown at runtime.

2. yield , sleep are static members of Thread class.
3. stop , resume , suspend are deprecated members of Thread class.
4. InterruptedException is checked exception.
5. Given
   synchronized (expression){…..}
   - expression may evaluate to reference to object.
   - expression cannot evaluate to primitive type or null.
   - If execution block completes (abruptly or normally)the lock is released.
   - synchronized statements can be nested.
   - synchronized statements with identical expression can be nested.

6. Thread.yeild method
   - may cause the current thread to move from runnable to ready state .
   - It isn't guaranteed i.e. same thread may start executing again.

7. Thread.sleep method
   - makes thread to move from running to not – runnable state
   - thread takes lock with it if it has got one.

8. Thread class inherits Runnable interface.
9. A dead thread can never be restarted.
10. When overriding the 'run' method of the Thread class remember the following points
    - the prototype must be public void run(){}
    - Access modifier must be public.

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

- ReturnType must be void.
- No arguments must be specified and if done it doesn't cause error but it can't be used to start a new thread.

→ if the class implements Runnable interface then it has to possess run method with no arguments specified otherwise compiler error is caused.

- it must not declare any new checked exception in its throws clause.
- it must not be declared static
- It can declare unchecked exception in its throws clause.
- If start method is called twice then illegalThreadStateException is thrown.

11. the run method in Thread class is empty method which does nothing.
12. join,sleep,wait methods declare InterruptedException in their throws clause.
13. If we extend thread class or implement runnable interface then the present run() method must not be marked static ,if it is marked it causes compiler error.
14. The priority of a thread can be altered after the thread has been started using setPriority method .The code compiles fine.

Fundamental Classes in the java.lang Package

1. Wrappper classes :-
   - Wrapper classes are immutable.
   - given,
     Float f = new Float("Str");

Str can take values	Str can't take values
123 or +123 i.e. positive integer(with '+' sign)	1l or 1L i.e. suffixing 'L' will cause error.
-2343 i.e. negative integer	0x10 i.e. we cannot give hexadecimal numbers as input.
±1.0 or ±.0d i.e. double number	
±.0f i.e. float number	
±1.1e5f i.e. we can have 'e' in the input string	
±1.1e5d i.e. we can have 'e' in the input string	

   - given ,
     Long l = new Long("Str");

Str can take values	Str can't take values
123 i.e. positive integer without '+' sign	+123 i.e. positive number with '+' sign
-234 i.e. negative integer	0x133 i.e. hexadecimal number
	±1.0f i.e. floating point

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

	number
	±1.0 i.e. double precision number
	±.1e1f or ±.1e2d i.e. 'e' is not allowed
	1l or 1L i.e. suffixing 'L' is not allowed

- Float constructor is overloaded,
  - takes a primitive "double number"
  - takes a string
  - takes a primitive "float number"

- The equals method when used with wrapper classes gives output as true if
  - both input objects are of the same class
  - both have the same value

  → Remember if two objects of different class are input to equals method then the output is false and not compiler or runtime error.

- there isn't a no argument constructor in any of the wrapper classes.
- given,
  Byte b = new Byte(1);//causes compiler error because 1 → is an integer and Byte constructor takes only byte .similarly for short.

- given,
  Long lg1 = new Long(1);
  Long lg2 = new Long(lg1);//causes compiler error because Long constructor doesn't take in Long object
- <wrapperClassName>.toString(); is overloaded
  - takes a no argument constructor and returns string of the given wrpper object
  - (static)takes argument corresponding to the wrapper class i.e. Integer.toString() takes int number and Float.toString() takes float number only unlike its constructor
- wrapper classes (excluding Character and Boolean) extends Number class and has 6 methods and they are
  - byteValue
  - shortValue
  - intValue
  - longValue
  - floatValue
  - doubleValue

  → there isn't a "charValue" int the wrapper class(excluding Character).

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

- Boolean.valueOf() returns a reference to either Boolean.TRUE or Boolean.FALSE

  Boolean b1 = Boolean.valueOf(true);
  Boolean b2 = Boolean.valueOf("TrUe");

  "b1==b2" is true.
- Remember NaN is not equal to Nan.
- Boolean has two constructors,
  - takes boolean values i.e. true or false.
  - takes String values i.e given,
    Boolean b = new Boolean("TrUe");//input string if passes the equalsIgnoreCase method with the string "true" then the new Boolean object will contain value - true.
  - given,
    Boolean b = new Boolean(<<AnyStringLiteral>>);
    for e.g. Boolean b = new Boolean("santosh");
    Boolean b1 = new Boolean(null);
    then , b.equals(b1) will be true because b and b1 contain – false .
- given,
  Double a = new Double(Double.NaN);
  Double b = new Double(Double.NaN);
  then a.equals(b) will be true.
- Given,
  Double c = new Double(0.0);
  Double d = new Double(-0.0);
  c.equals(d) → gives the output of false.
- Remember "0.0 == -0.0" is true.

2. **String class –**
   - It is final.
   - String objects are immutable.
     Not methods of this class –append, delete, insert. Hence a code fragment like this causes compiler error ,
     String s = "A";
     s.trim();s.append();//causes compiler error since append is not String method
   - "valueOf" is static method of String class.
   - String has many constructors ,some of the important ones take input parameters
     - string
     - stringBuffer
     - byte array
     - char array
     - Doesn't take in any argument.

     → String constructor doesn't take in parameter such as char, byte, int, short,

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

int array ,any array other than char array and byte array etc .

- String class has a method called length() where as array has attribute called 'length'
  hence, String s = "abcd";
  int a = s.length;//causes error
  similarlay
  int [] a = {1,2,3,4,5};
  int b = a.length();//causes error
- String s1 = new String("abcd");
  String s2 = s1.toString();
  here toString method returns the reference to the existing object hence "s1==s2" will be true.
  Note: Even substring(),trim(),concat(),return the same string if they don't modify the given string.
- String s1 = "santosh";
  String s2 = "santosh";
  then "s1==s2" is true ,since while creating a string object the compiler looks for the same String in the pool and if it finds one then it returns the reference to the existing string.
  Note: given ,
  String s ,s1 ,s2,s3;
  s=s1=s2=s3=null;
  {….compelx code….}
  s=s1+s2;
  s3 = s1+s2;
  then "s == s3" will return false.

3. **StringBuffer class**
   - StringBuffer objects are mutable.
   - StringBuffer class doesn't override "equals" method .
     Hence two different objects are equal if and only if the two references are referring to the same object i.e. it executes the "==" operator on the two references inside the equals method.
   - The setLength() method of StringBuffer class sets the length of the StringBuffer object to the specified integer number passed as the parameter to the method . If the number is less than its present length then it truncates the StringBuffer.for e.g.

     StringBuffer s = new StringBuffer("santosh");
     s.setLength(3);
     System.out.println(s); //prints "san"

   - Since StringBuffer class doesn't override "equals" method it doesn't override "hashCode()" method also.
     Hence the hashCode of two StringBuffer objects are equal if only they pass the equals method i.e. the two references must be pointing to the same

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

object. So ,if two StringBuffer objects pass the equals method then the posses the same hashCode value.

4. **Math class**
   - Math class is final.
   - All methods in math class are static.
   - It has only private constructors i.e. it can't be instatantiated and also cannot be extended.
   - There is no mod function in Math class.
   - Remember the prototype of the major functions in Math class.
     - public static strictfp double abs(double);
     - public static strictfp int abs(int);
     - public static strictfp long abs(long);
     - public static strictfp float abs(float);
     - public static strictfp double max(double,double);
     - public static strictfp float max(float,float);
     - public static strictfp long max(long,long);
     - public static strictfp int max(int,int);
     - public static strictfp float min(float,float);
     - public static strictfp int min(int,int);
     - public static strictfp long min(long,long);
     - public static strictfp double min(double,double);
     - public static strictfp long round(double);
     - public static strictfp int round(float);
     - public static strictfp double ceil(double);
     - public static strictfp double cos(double);
     - public static strictfp double floor(double);
     - public static strictfp double sin(double);
     - public static strictfp double sqrt(double);
     - public static strictfp double tan(double);
     - public static strictfp double toDegrees(double);
     - public static strictfp double toRadians(double);
     - public static strictfp double random();

   - Math.random() takes no argument(seed) and returns a double value lying between 0 and 1 .Furthur the value returned can be equal or greater than 0 while it must always be less than 1 i.e.
     $0.0 \leq$ Math.random < 1.0
   - No methods decare non-primitive return type or non-primitive argument type.
   - Math.abs method,
     - Math.abs(-2.4/0.0)= +Infinity
     - Math.abs(2.4/0.0) = - Infinity
     - Math.abs(Integer.MIN_VALUE) = Integer.MIN_VALUE;
     - Math.abs(Long.MIN_VALUE) = Long.MIN_VALUE;
     - Math.abs(Float.NaN) = NaN;
     - Math.abs(-0) = 0;

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

Note: It can't return Negative infinity and Negative Zero.

- If we pass a value lying between -1 and 0 to Math.ceil method then we get -0.0 as the output.
- If we pass -0.0 and +0.0 as the arguments to max or min method we get the foolowing results,
  - Max method – return value is 0.0
  - Min method – return value is -0.0

5. **Collection class**
   - Given,
     Object a = new LinkedHashSet();
     Then a instanceof HashSet is true because linkedHashSet extends HashSet.
   - We get similar results with LinkedHashMap and HashMap since LinkedHashMap extends HashMap.
   - Remember these points
     - Collection → interface
     - Collections and Arrays→ classes providing utility methods for various operations.
   - ListIterator is an interface which extends only Iterator interface not List interface
   - Enumeration interface was introduced in java 1.0
   - Iterator was added in java 1.2
   - Iterator methods : -
     - hasNext();
     - next();
     - remove(Object);
   - Enumeration interface has methods
     - hasMoreElements();
     - nextElement();
   - the default number of rows visible in a List is '4'.
   - When we add the same element again into a Set using add(Object) then the subsequent additions of the element returns a boolean false in its reurn part.

Operators and assignments:-

1. given,
   class A{}
   class B extends A{}
   B b = null;
   A a = b;
   B b1= (B)a;
   The above code fragment doesn't cause NullPointerException since we are casting amongst the members of inheritance tree. Hence b1 will have the value "null".
   Similaraly,
   given,

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

```
int[] i = null;
Cloneable c = i;
i = (int [] ) c;
```

The above code fragment doesn't result in runtime error(NullPointerException) since any array extends Cloneable interface.

2. All arrays implement Cloneable interface and java.io.Serializable interface
3. given,

```
long [] l = {1,2,3,4,5};
int [] i  = {10,20,30};
l[0]= i[0];//legal and correct
l = i; //causes compiler error
l = (long[])i;//causes error
```

4. given,

```
final short s1 = 1;
final char c1 = 2;
short s2 = 3;
char c2 = 4;
byte b1,b2,b3,b4;
```

remember the following,

```
b1 = s1;//legal because compiler applies implicit casting since it is final variable.
b2 = c1;//legal because compiler applies implicit casting since it is final variable.
b3 = s2;//illegal because compiler doesn't apply implicit casting.
b4 = c2;//illegal because compiler doesn't apply implicit casting.
```

5. Important formula,

```
-x = ~x +1;
```

and

```
~x = -x-1;
```

6. if ,

```
int i1;
int i2;
int i3;
i1 = <<someNumber>>;
i2 = <<someComplexMethodName>>();
i3 = i1<<i2;
```

if i2 is greater than 31 then "i2&0x1f" is used in its place.

7. The assignment operators are,

=	+=	-=
*=	/=	%=
<<=	>>=	>>>=
&=	^=	|=

Note:
There is no assignment operator like &&=, ||= , ~=.

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

8. given,
   byte b = 1;
   long l = 1000;
   b += l;//line 3
   the above fragment will not cause compiler error since compiler applies implicit casting before executing the line number 3.the above code fragment will be executed as follows.
   - b += l;
   - b = (byte)(b+l);
   - b = (byte(1+1000);
   - b = someValue;

   In general ,
   E1 op= E2;
   can be written as E1 = (type)(E1 op E2);
   here type is the datatype of the operand E1 and "op" is the operator applied.

given,
interface inter{ }
class B implements inter { }
class sub implements B { }

```
class Test
{
    public static void main(String [] args)
    {
            B b = new B();
            I i = b;
            sub s = (sub)b;//line 1
    }}
```

9. At line one compiler error doesn't occur because we applying explicit casting on members of the same inheritance tree. Thus at line 1 runtime error and not compiler error is caused. This is be cause B is superclass of sub and an instance of B cannot be converted into a type of sub class.

```
class Test2
{
        public static void main(String [] args)
        {
                B[] b = {new B()};
                sub[] s = new sub[1];
                Object obj = b;
                s = (sub[])obj;//line 1
        }
}
```

10. At line one a runtime error and not compile time error is caused.

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

11. given,
    byte a = 1;
    byte b = 2;
    byte c = (byte)a+b;// Line 1
    line 1 causes compiler error since casting is done only to a and not to sum of a and b.As a result the RHS will be int and LHS will be byte which results in compiler error.

General points:

1. Evaluation and execution –remember that evaluation is from left to right but execution is from right to left.
2. there must be some statement after do keyword in do – while loop for it to compile without error.
   i.e.
   do ; while(false); //correct
   do {;}while(false); //correct
   do {}while(false); //correct

   do while(false); //error
3. If "t" is a reference variable then ,
   t.equals(null) → is false
   (null).equals(t) → compiler error
   let's say t2 is some other reference variable then
   t.equals(t2) → false and not error
   consider,
   t = null;
   t.equals(t2);//not compiler error but runtime error
4. If a class is declared inside a package with public modifier then that class becomes invisible to all other classes in other packages unless they import the package or use extended form of addressing the class.

Q What can stop a thread from execution?
A → Program exiting via call to System.exit(int);
→ Another thread's priority is increased
→ A call to stop method in Thread class.

Q What will stop a thread from execution?
A → Program exiting via call to System.exit(int);
→ A call to stop method in Thread class.

5. The Iterator method of collection interface when invoked returns an instance of Iterator class.
6. given,
   char c = 'a';
   int i = 1;
   c + =i; //correct
   c = c+ i; //illegal

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

7. when use int numbers in basic arithmetic operation then the output is an integer number. Hence ,
   int i = 4/3;
   "i" will have the value 1.
8. native methods can be set to any access level - public , protected, private, default.
9. The methods in the immediate super class in the inheritance tree may be accessed through the use of keyword "super" , but classes above the immediate super class are not visible.
10. valid comments
    - /* this is a comment */
    - /** this is a comment */
    - /*this is a comment **/
    - // /** this is a comment */ */

    Important:-
    - /* //this is a comment */
11. invalid comments
    - /** this is a comment */ */

12. If a method declares some exception in throws clause and the subclass of the given class while overriding the method declares some new exception then before assuming that it causes compiler error first check whether the new exception thrown in the subclass' method is unchecked exception or not.
13. After solving the logic inside the problem before jumping to conclusion check whether some code is unreachable or not. Because if it happens so then it results in compiler error.
14. the following form of instantiating a static inner class results in compiler error,
    new <outerClassName>().new <staticInnerClassName>();
15. long l = Integer.MAX_VALUE;
    float f = l;
    double d = l;
    then "d==f" is false due to rounding of numbers in float literal.

    But when we assign l to Long.MAX_VALUE or to Integer.MIN_VALUE then we get the result of "d==f" as true.
16. we can place label statements around a block of code wherever we wish , unless the name used is not a keyword and follows all the rules meant for identifier.
    For eg.

```
labelA:
{
        …..some complex code….
        …..some complex code….
        if(someThingIsTrue)
        {
                break labelA;
        }
}
```


Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

this way we place break statement with label in any labeled block of code which may break out of the code if something comes true.

Furthur the same labels can be used for other block of code as long as they don't overlap.

17. An abstract method cannot be marked as both
    - Abstract and strictfp
    - Abstract and native
    - Abstract and synchronized
    - Abstract and final
    - Abstract and private
    - Abstract and static
18. shift operators can be used only on integers .
19. switch statements can evaluate byte , short, char , int.
    but not long, float.double.
    i.e. long l = 10;
    switch(l){ }//causes compiler error

    → before jumping to conclusion about switch statements , verify whether the case arguments are lying within the range of the switch argument.
    For e.g. byte b = 10;

    ```
    Switch(b)
    {
        case 10: …….. complexcode……break;
        case 1000: …….. complexcode……break;
    }
    ```

    here second case statement causes compiler error since it is out of range of byte literal.
20. the case argument must be primitive literal type or final variable.<compatible type>
21. for loop declarations,
    valid

    - for(int i=0,j=0;i<10;i++);
      i=10
    - for(i=0,j=0;;);
    - for(;;);

    invalid

    - for(i=0,int j=0;;);
    - int k =1;
      for(int i=0,k=0;;);

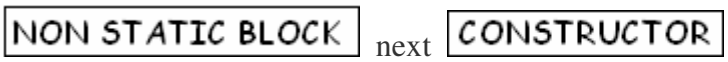
Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

22. equals(),toString(),hashCode() are public in Object class.
23. If a nested class has static nested variable then it can be used without instatiating the inner class.
24. Unicode representation of char is hexadecimal form of representation.
25. If we place a method declaring void as its return type in any assignment or print statements ,it results in compiler error.

26. If there are any static blocks in a class then when the class is first loaded i.e. calling its static method or instantiating it for the first time ,these blocks of code will be executed in order in which they appear in the declaration of the class. Further if there are other non – static blocks then they are executed before running the constructors. Summarizing,
    First time loading,

    during the subsequent instantiation of the class the non – static block followed by the constructor is executed.
    subsequent instantiating of the class,

27. when we invoke a static method of a class then , thread will obtain the lock of the instance of the given class in java.lang.Class created at runtime by the JVM. When we need to call the wait ,notify ,notifyAll method then we need to obtain the reference of that object in the java.lang.Class . to do this we use the form, <className>.class .wait();

28. we can apply final modifier along with transcient , static but not volatile but we can apply both transcient and volatile modifiers to field at the same time.
29. let a method by name m1() calls another method by name m2() which again calls another method called m3(). Let's say m3() throws a smaller exception ,then method m2 can declare a broader exception than the one passed from m1 or declare the same exception or it can even put in a try-catch block i.e. to say as we go down the call stack methods can declare wider exceptions in their throws clause.
30. given,
    class A extends B{} //Line 1
    class B extends A{} //Line 2//causes compiler error
    here at line 1 compiler error is not caused but instead at line 2 compiler complains of cyclic inheritance.

31. All the Unicode characters have four numbers (hexadecimal) placed after '\u

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

i.e. '\uXXXX'
here X → can take the values 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f
hence the following results in compiler error
'\u002','\u030'

32. In some questions though it may seem that we are initializing a variable before using it through a conditional block (like having only if – statement and initializing a variable before using it)but instead it will be certain that the variable will be initialized (like having a else statement along with if statement).In some cases even default statement in switch construct carries out similar function.
33. within a method local inner clss the final variables of the enclosing method can be accessed but cannot be assigned a value even if they aren't previously assigned a value. → this results in compiler error.
34. If an exception is thrown again from a catch block then before exiting the finally block will be executed where the exception may be made to be discarded by placing a return statement.for example,

```
……..main(String [] args)
{
        try{…some complex code….}
        catch(ArithmeticException e){throw new Exception();}
        finally{return;}
}
```

this prints nothing .

35. variables declared in a try block are local to that block and aren't even visible in the catch ,finally or any other block of the same method.

```
try
{
        int i=2;
        throw new Exception();
}
catch(Exception e)
{
        …print(i); // causes error since 'i is not visible in the catch block.
}
```

36. following statements cause compiler error,

```
//  char a = '\u000a';
 //  char b = '\u00b';
/* char c ='\u00d'; */
```

the below statement doesn't cause error

```
/* char d = '\u000a'; */
```

37. the JVM exits only when
    - All non – daemon threads complete
    - Some thread calls System.exit(int) method

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

38. A class declared in an interface
    - Is always public
    - Is always static
    - Cannot access i.e. the methods declared in the class cannot call other methods declared in the interface.
    - Can have final, abstract, modifier.
39. consider,
    String str = null;
    Here str behaves in all the assignment and print statements in the same way as it would behave if it had contained the string "null".

40. Even if the modifier applied to class is public , if the constructor is placed explicitly by the programmer without any modifier then it won't have the same access modifier as the class but instead it would possess package access.
    - Hence classes outside the given package can't instantiate this class.
    - Hence classes outside the package can't extend this class, since the constructor will not be visible to them.
41. consider the following conditions,
    - we have an interface containing method which throws some exception in throws clause.
    - Another class implements this interface and doesn't declare any exception in it's throws clause.
    - Now we instantiate the class using the interface reference variable.
    - When we make a call to the method of the interface using the interface reference then we have to declare the exception in throws clause or put it in try – catch block.
      For e.g.

```
interface I{
void m()throws java.io.IOException;
}

class A implements I{
void m(){}
}

class B{
….main(String[] args)
{
    I i = new A();
    i.m(); //causes compiler error → IOException must be placed in try-catch block or declared in throws clause
}
}
```

42. Let's say we have a static final public variable in one class and is accessed by another class. Then after compilation of both the classes ,if the previous class's

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

final variable's value is changed and again compiled (but the second class remains unchanged).Now when the second class is executed then it will have the original value and not the revised value .For e.g.

```
public class A{
        public static final int i = 10;
}

public class B{
……..main(Atring [] args)
{
……..print(A.i);
}
}
```

when class A and class B files are compiled then we will get two class files named A.class and B.class and when B.class is executed we get the output of "10".

Later the value of i in class A is changed to 100 and again compiled. But class B is maintained as it is i.e. it is not compiled again. Now we will have two class files named A.class(changed one) and B.class(old one). class A now contains variable i with value 100 but when we execute B.class we get the output as 10 again instead of the expected 100.

If we remove the final modifier for variable i in class A we get the output of 100 during the second compilation of the class B.

43. If a variable is declared and initialized inside a non – static or static block or inside a constructor it becomes inaccessible to all other methods within the class excluding that block
44. If we send NaN as one of the arguments for Math.min or Math.max we get the output as NaN.
45. when we use instanceof operator the left operand must be an instance of Object class or an instance of its subclass The right operand must be a class , Interface type.

    If the object is an instance of thegiven class or one of its subclass then it returns true otherwise it returns false.
    new String() instanceof Object → true
    new Object() instanceof String → false

    Here the compiler won't complain as long as the left operand belongs to the inheritance tree including the right operand otherwise it causes compiler error.

46. float f = Float.MIN_VALUE /Float.MAX_VALUE;
    then the value of "f" is 0.0f

47. toString method of wrapper clsses can be invoked in the following form,
    Integer.toString(Number,radix);
    Here
    Radix is usually 2,8,10,16 but it can take values ranging from 0 to 36.
48. Arrays must be indexed with the values of the type byte, short ,char or int but not long ,float or double.
49. "this" reference is final so any new reference cannot be assigned to it.→ results in compiler error
50. the exception thrown by finalized method is ignored by JVM
51. Initializer blocks(both non –static and static blocks) cannot throw exceptions
52. important points to be noted while using modulus operator –
    - anything nodulus 0.0 or -0.0 is NaN and not ±Infinity.
    - Anything divide by 0.0 is +Infinity
    - Anything divide by -0.0 is –Infinity
    - ±0.0 divide by ±0.0 is NaN.
    - 0.0 divide by anything is 0.0
    - -0.0 divide by anything is -0.0

53. given,
    class A{void m(){}}
    class B{}
    interface I {}
    then considedr the followingm
    A a = null;
    B b = null;
    I i = null;
    b = (B)a; //compiler error since we can't cast B into A as they don't belong to same inheritance tree.

    I = (I)a;//No compiler error but runtime error results due to this.

    Now,
    i.m();//compiler error because method m() is invisible to reference variable of interface I

    but ,
    ((A)i).m();//normal compilation but run time error(ClassCastException).
54. Proper instantiation of non – static inner class given,
    class Outer
    {
        class Inner{}
    }

    Outer.Inner name = new Outer().new Inner();

    OR

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u

Outer o = new Outer();
Outer.Inner name = o.new Inner();

The following instantiation compiles fine but results in runtime error,

Outer o = null;

Outer.Inner name = o.new Inner(); //run time error(NullPointerException) is caused

55. Look for programs where keywords are used as labels for many blocks and loops. This causes compiler error.

For e.g.

```
……..main(String [] args)
{
        int i = 2;
        case:for(i=0;i<10;i++)
        {
                if(i==5)break case; //causes compiler error
                System.out.println(i);
        }
}
```

56. If a class implements an interface and doesn't provide all the required implementing methods then compiler error is caused at the first statement of the class where we would have said that the class implements the interface
For e.g.

```
public class A implements Runnable //compiler error is caused at this line
{
        public void run(int i){}
}
```

57. If a thread is blocked in the wait method of an Object and another thread executes notify method on the same object then the thread might never start at all.
58. static members of static inner classes can be referenced by classname of static inner class like,
<OuterClassName>.<StaticInnerClassName>.VariableName → this variable is declared static.

Anand Bheemarajaiah(anand_bheemaraju@yahoo.com)
www.geocities.com/sunjava4u