```
JAVA NOTES BY JOHN GRIFFITHS (C) 2007 RED91.COM
-----------------------------------------------

javac -version
(current version installed)

javac [source file]
(compile [source].java file to .class file)

java [compiled file]
(run [binary].class file)

regular expressions
\n = newline

byte    (8 byte binary)
short   (short integer)
int     (integer)
String  (character)
boolean (true or false)
float   (floating point integer, 32bit)
double  (double byte floating integer, 64bt)

primitive types:
boolean, char, byte, short, int, long, float, double

String[] s ("11", "22")
(create string array)
(use s.length to determine number of elements, starts at 1)

int[] i (1, 2)
(create int array)
(use i.length to determine number of elements, starts at 1)

...arrays are objects too
int[] nums;
nums = new int[7];


if () {} else {}
while () {}
for (int x = 0; x < 10; x = x + 1) {}

for (String name : nameArray) {}
    // enhanced for loop, fires for every element in name array
    // new in Java 5.0

Dog mydog = new Dog();  <- create new reference to object
mydog.play();   <- run method play of class mydog


==  compare
=   assign
<   less than
>   greater than
>=  greater or equals
<=  less than or equals
||  or
&&  and
!=  not equals
!   not like

[int]++  add 1
[int]--  remove 1

GC = Garbage Collector

System.out.print() <- print text
System.out.println() <- print line


class [name] {
    instance variables
    (cannot put code here)

    void [methods]() {
        [behavior]
        (code goes here)
    }

    public static void main (variable args) {
        (startup code)
    }

}

[class] classinstance = new [class]();
- create new class instance

[return type] methodname(method parameters) {
    //behavior commands
    return [value]
}


main - used to start the application, init app
-> public static void main (String args) {}

... args) <- arguments required

public (global, available outside class)
private (only available within class)
protected (cannot be extended via inheritence, etc.)

static (cannot be changed, set in stone)
void (no values returned / required)

instance variables are declared inside a class but not within a method.
local variables are declared within a method and need a value assigned.


Integer.parseInt("3") //converts string to integer


ArrayList
---------
import java.util.ArrayList; //put at start, before you use it
    // needed to give us the ArrayList type
    // must declare class library fully anywhere you intend to use it
    methods :
        .add(object)
        .remove(object)
        .contains(object elem: returns true if elem in ArrayList)
        .isEmpty(returns true if empty)
        .indexOf(object elem: returns where elem located)
        .size(returns num of objects in ArrayList)
        .get(int index: returns object at current position)


[object] = null;
// destroy object, set it to null

...constructor
// a method that has same name as class
// e.g.
//  class Dog {
//      public Dog() {...}
//  }
// ...used to initialise the class elements

...destructor
// used to destroy / clean up after the class is of no use


...inheritance
class [classname] extends [prior class] {}
// means one class can inherit the instances & methods from another class
// if you overwrite a method you can get at it using super.[old method];


...overloading a method
// having two methods with the same name but different argument lists
// e.g.
// public int addNums(int a);
// public double addNums(double a);
// -- note: can't change just the return type!, have to change both


public abstract void eat();
// [abstract] means nobody can make a new instance for that class
// doesn't have a body, just ends in a ;
// you just get to declare it like above (e.g. like a library item)
// an abstract method must live inside an abstract class
// however you can build upon it by extending it from another class


polymorphism
// ...means an object with many forms


static & non-static classes
// a static class doesn't have any instance variables that can affect
// it's operation.  a non-static one does.
// if you try to use an instance variable inside a static method you'll
// get an error.
//  -- static's can't see instance variable states
//
// if you want to use one inside it, use :
//  Static [type] [variable name] = [value];
//
// static variables are shared
// instance variables have one per object


final...
// final class name {}      final classes cannot be extended
// final void method() {}   final methods cannot be overridden


super.[method]
// when you override a method, you can still get the original method
// by using super.thatmethod()
// e.g.
//  you create a new class that's based on another (inherited)
//  but you want to use the original method and not the inherited one,
//  so you use the super.[method] to call it
//
//  super(thing);       call the original constructor
//  super.mymethod();   call the original method


math methods
// Math.random()        returns a random number
// Math.abs(num)        returns absolute value (-240 => 240, 11 => 11)
// Math.round(num)      rounds double to int (11.7 => 12)
// Math.min(num, num)   returns min value (22, 11 => 11)
// Math.max(num, num)   returns max value (22, 11 => 22)


// number formatting
.format("%, d", 1000000);
    %, d    format as integer + commas -> 100,000,000
    %.2f    format as floating + 2 point -> 100000.00
    %,.2f   format as floating + commas + 2 point -> 100,000.00
    %x      format as hex -> 2a
    %c      format as character code -> 42 = * (ascii code for *)


// date & time formatting
.format("%tc", new Date());
    %tc     format as complete date & time -> sun nov 28 12:11:42
    %tr     format as time only -> 12:11:42
    %tA, %tB, %td   format separately
    // Date today = new Date();


wrapping a primitive
// int i = 288;                       // integer
// Integer iWrap = new Integer(i);  // wrapping the integer into an object
// int iu = iWrap.intValue();       // unwrapping the object
```

```
exception class hierarchy
//
//         Throwable
//             |
//         Exception
//             |
//     |----------------|
//     |                |
// IOException  InterruptedException
//
// try {do risky thing}
// catch(Exception ex) {try to recover}
// finally {cleanup code}
//
// when you have multiple catch block, arrange them in order, putting
// smaller ones at the top and large ones at the bottom.
//
// catch([exception method] ex) {recovery code}
//           |-- relates to a method you can create to throw it right
//
//
//  public void foo() throws ClothingException { do.laundry(); }
//
//      then...
//
//  try { foo(); }
//  catch (ClothingException ex) { System.out.println("error!"); }
//  finally { System.out.println("the end..."); }


hash tables
// hash tables are a form of associative array.  it's like an abstract
// data type that has a collection of keys and a collection of values.
// so the value associated to "bob" is 7, ("bob" is mapped to 7)
//
//  operations usually used:
//      add
//      re-assign
//      remove
//      lookup
//
//  need: import java.util.HashMap;
//
//  HashMap<String, String> phoneBook = new HashMap<String, String>();
//  phoneBook.put("sally smart", "555-9999");
//
// why?
// best used in memoization;


[object].hashCode();        // returns unique id for object
[object].equals(object);    // tells if two objects are considered equal
[object].getClass();        // gives back class object was created from

[object].toString();        // returns name & value (object to primitive)
[object].intValue();        // return integer (object to primitive)
[object].booleanValue();    // return boolean (object to primitive)


serialization
// this allows your java app to flatten the objects into something that
// can be stored in a file and read by the computer, not a human.
// so you can restore session objects into memory with no fuss.
//


gui
// import javax.swing.*;        for the display, boxes, buttons (out)
// import java.awt.event.*;     for the ActionListener & ActionEvent (in)
//
// swing components...
//      JButton, JCheckBox, JTextField, JFrame, JPanel
//
// three layout managers...
//      BorderLayout, FlowLayout, BoxLayout
//
```

```
//      panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));


serialization...
// e.g. objects can be flattened and inflated
// allows you to save whole objects to disk and read them back intact.
// allows you to save the state of an object
//
// use if..     your data will only be used by the program that created it
// don't if..   your data will be used by other porgrams
//
// data moves in streams
//
//  class [name] implements Serializable {...}
//                                |----allows object to be saved
//
//      FileOutputStream(file)    <- writes stream to file
//      FileInputStream(file)    <- reads stream from file
//
// Serialization versioning
// ------------------------
// over time your class may change, new instances added, etc. and you
// may find that old object saved to file may be incompatible with the
// newer class, you can check this by creating a serialVersionUID in
// your class...
//
// at the command line type...
// john-griffiths-ibook-g4:~/java indiehead$ serialver JFile
//
// this will return...
// JFile:    static final long serialVersionUID = 1381561046643289825L;
//           --- paste this into your class as an instance


Networking
// you need a ip address and port number...
//      Socket chatSocket = new Socket("196.164.1.103", 5000);
//
// to read data from a socket...
//      1. make a Socket connection
//      2. make an InputStreamReader chained to a low-level input stream
//      3. make a BufferedReader and read
//
// to write data to a socket...
//      1. make a Socket connection
//      2. make a PrintWriter chained to a low-level output stream
//      3. write (print) something
//
// import java.io.*
// import java.net.*


multi-threading
// this is built into java, basically allows you to run many connections
// concurrently at the same time.
//
//  create a thread...
//      Thread t = new Thread();
//      t.start();
//
// java has multiple threads but one thread class.
//
//  how to launch a thread...
//      1. Runnable threadJob = new MyRunnable();
//      2. Thread myThread = new Thread(threadJob);
//      3. myThread.start();
//
// every thread needs a job to do.
//      e.g. a method to put on the thread stack
//          public void run() { code to run in the thread }
//
// put a thread to sleep with...
//      Thread.sleep(2000);
//          will knock a thread off the running stack for 2 seconds


File Buffers
```

```
// using buffers makes it easier to write to a file, because rather than
// having to save everything it's told; it'll only save once the buffer
// is full.  easier on the disk that way.
//
// you can tie a buffer to the FileWriter easily using...
//      BufferedWriter writer = new BufferedWriter(new FileWriter(aFile));
//
//      FileWriter  <- writes character streams rather than bytes
//
// however, if you don't want to wait, just flush the buffer...
//      writer.Flush();


...file & dir actions
// imports java.io.File;
//
//  File f = new File("myfile.txt");
//  -- create file represented by object
//
//  File dir = new File("newdir");
//  dir.mkdir();
//  -- create new directory 'newdir'
//
//  boolean isDeleted = f.delete();
//  -- delete file
//
//  System.out.println(dir.getAbsolutePath());
//  -- get absolute path of file or directory


string splitting
// you can split a string into many elements by using .split("/"); this
// will return a your string split into elements of an array which you
// can access like...
//
//  String[] tokens = lineToParse.split("/");
//  System.out.println(tokens[0]);  // print first element of string array


ArrayList and friends...
// ArrayList is a collection that can be used to store & create dynamic
// arrays of string, integers and class objects. however it's not just
// the only one...
//
// import java.util.*;
//
//  TreeSet       : keeps elements sorted & prevents duplicates
//  HashMap       : lets store elements as name/value pairs (hash table)
//  LinkedList    : designed to be better than ArrayList (but usually not)
//  HashSet       : prevents duplicates, given an element can find that
//                  element in the collection fast
//  LinkedHashSet : like a regular hash map, except it can remember the
//                  order of elements (name/value), where inserted &
//                  where last accessed
//
// collections possess the Collections.sort() method, sorts alphabetically


Generics (new to Java 5.0) + More Collections
// allow you to write type-safe collections, e.g. code that stops you from
// putting a Dog into a list of Ducks.
//
// 'extends' means 'extends or implements' depending on the type
//
// ArrayList<String> thisList = new ArrayList<String>
//  means...
// public class ArrayList<E> extends AbstractList<E> {
//  public boolean add(E o)
// }
//      E is element
//
// three main interfaces...
//      LIST    : when sequence matters
//      SET     : when uniqueness matters
//      MAP     : when finding something by key matters (hash tables...)
//
// HashSet<Song> songSet = new HashSet<Song>();
// songSet.addAll(songList);  <- takes one collection and populates the
//                              hashset in one go
//
// HashSet, can check for duplicates by using...
//      HashCode()  : unique id you can return from objects
//          return title.hashCode();
//      equals()    : called to check if two objects really the same
//          return getTitle().equals(s.getTitle());
//
// TreeSet, for sets that you want to stay sorted...
//      TreeSet elements must be Comparable.
//          class Book implements Comparable {...}
//
// Map, a collection that acts like a property list
//      HashMap<String, Integer> scores = new HashMap<String, Integer>();
//      scores.put("kathy", 42);
//
// .... remember -> for(Animal a: animals) { a.eat(); }
//
//  (ArrayList<? extends Animal>)
//      ? = wildcards
//          e.g. you can do things with the list elements, but you can't
//          put new things in the list.
```

JAR files & deployment

```
// to put your code into a JAR file, first put your class files into
// the /classes directory, like below...
//
// javac -d ../classes MyApp.class
//       -d means create the class files and put them in a directory next
//          to the one your already in.
//
// now create a manifest.txt file, that states which class has the main()
// method...
//      Main-Class: MyApp  (minus .class)
//
// run the JAR tool to create a JAR file that contains everything in the
// class directory, plus the manifest
//      cd myapp/classes
//      jar -cvmf manifest.txt appl.jar *.class
//
// or just one...
//      jar -cvmf manifest.txt appl.jar myapp.class
//
// executing the JAR...
//      java -jar appl.jar
//
// putting your class into a package reduces the chances of naming
// conflicts...
//      reverse domain package names...
//          com.myjava.projects.Chart    <- the class name is always
//                                          capitalized
// 1. choose a package name -> com.myjava
// 2. put a package statement in your class...
//          package com.myjava;  <- at the start of your .java file
// 3. setup a matching directory structure...
//    -->  myjava/classes/com/myjava
//                /source/com/myjava
//
// to compile a .java into a package...
//      cd myapp/source
//      javac -d ../classes com/myjava/package.java
//
// to compile them all...
//      javac -d ../classes com/myjava/*.java
//
// to run the package...
//      cd myapp.classes
//      java com.myapp.package
//
// to make an executable JAR with packages...
//  1. make sure all your class files are within the correct dir structure
//     and under the classes dir.
//  2. create a manifest.txt file that states which class has the main()
//     method, and be sure to use the fully-qualified class name...
//          Main-Class: com.myjava.package
//  3. build with...
//          cd myapp/classes
//          jar -cvmf manifest.txt packEx.jar com
//                  all you specify is the com dir, and you'll get
//                  everything in it.
//
// list contents of a JAR...
//      jar -tf packEx.jar
//
// extract the contents of a JAR (unjar)...
//      jar -xf packEx.jar
//          -xf = extract file
```

Java Web Start

```
// this allows clients (web pages) to run your jar files via a jnlp file.
// how...
//  1. make an executable JAR for your application
//
//  2. write a .jnlp file
//      a .jnlp is a form of xml file, with a <resources> element that
//      points to your .jar file
//
//  3. place your jar + jnlp files on the server
//
//  4. add a new mime type to your web server...
//      application/x-java-jnlp-file
//
//  5. create a web page with a link to your .jnlp file
//      <html>
//          <body>
//              <a href="myapp.jnlp">launch app</a>
//          </body>
//      </html>
```

Remote Method Invocation technology (RMI)

```
// method calls are always between two objects on the same heap, however
// RMI allows you to invoke a method on an object running on another
// machine.
// in RMI the client helper is a stub and the server is a skeleton
//
// Making a remote service...
//  1. make a remote interface
//      public interface MyRemote extends Remote {} ...
//  2. make a remote implementation
//  3. generate the stubs and skeletons using rmic
//  4. start the rmi registry (rmiregisty)
//  5. start the remote service
//
//  be sure each machine has the class files it needs
```

Servlets

```
// are programs that run on (and with) an HTTP web server.  when a client
// uses a web browser to interace with a web page, a request is sent back
// to the web server.  if the request needs the help of a java servlet,
// the web server runs (or calls) the servlet code.
//
// a simple servlet...
//
import java.io.*;
import javax.servlet.*;
import javax.servlet.http;
public class MyServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse
                    response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String message = "message";
        out.println("<html><body>");
        out.println(message);
        out.println("</body></html>");
        out.close();
        }
}
```

```
Enterprise Java Beans (EJB)
// in Java 2 Enterprise Edition (J2EE) you get a web server and an
// enterprise java beans (EJB) server, so you can deploy an application
// that includes both servlets and EJBs.
// an EJB server steps into the middle of an RMI call and layers in all
// of the services (transactions, security, dba, networking, etc.).

Jini
// basically RMI with wings, it uses RMI and gives you this too..
//
//  1. adaptive discovery
//      a client needs something but doesn't know where to find it so asks
//      the lookup service, the lookup service responds, since it does
//      have something registered as the interface the client needs.
//
//  2. self-healing networks
//      a jini service registers with a lookup service, the lookup service
//      gives it a lease which has to be renewed over time, giving the
//      service an accurate picture of the services available to the
//      network.  if the service goes offline (somebody shuts it down)
//      then the lease can't be renewed and the lookup service drops it.
```