THE JAVA LANGUAGE CHEAT SHEET

Primitive Types:

INTEGER: byte(8bit),short(16bit),int(32bit),
long(64bit),**DECIM:**float(32bit),double(64bit)
,**OTHER:** boolean(1bit), char (Unicode)
HEX:0x1AF,**BINARY:**0b00101,**LONG:**8888888888888**L**
CHAR EXAMPLES: 'a','\n','\t','\'','\\','\"'

Primitive Operators

Assignment Operator: = (ex: int a=5,b=3;)
Binary Operators (two arguments): + - * / %
Unary Operators: + - ++ --
Boolean Not Operator (Unary): !
Boolean Binary: == != > >= < <=
Boolean Binary Only: && ||
Bitwise Operators: ~ & ^ | << >> >>>
Ternary Operator: bool?valtrue:valfalse;

Casting, Conversion

```
int x = (int)5.5; //works for numeric types
int x = Integer.parseInt("123");
float y = Float.parseFloat("1.5");
int x = Integer.parseInt("7A",16); //fromHex
String hex = Integer.toString(99,16);//toHex
//Previous lines work w/ binary, other bases
```

java.util.Scanner, input, output

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt(); //stops at whitespace
String line = sc.nextLine(); //whole line
System.out.println("bla"); //stdout
System.err.print("bla"); //stderr,no newline
```

java.lang.Number types

```
Integer x = 5; double y = x.doubleValue();
double y = (double)x.intValue();
//Many other methods for Long, Double, etc
```

java.lang.String Methods

```
//Operator +, e.g. "fat"+"cat" -> "fatcat"
boolean equals(String other);
int length();
char charAt(int i);
String substring(int i, int j); //j not incl
boolean contains(String sub);
boolean startsWith(String pre);
boolean endsWith(String post);
int indexOf(String p); //-1 if not found
int indexOf(String p, int i); //start at i
int compareTo(String t);
//"a".compareTo("b") -> -1
String replaceAll(String str, String find);
String[] split(String delim);
```

StringBuffer, StringBuilder

StringBuffer is synchronized StringBuilder
(Use StringBuilder unless multithreaded)
Use the .apend(xyz) methods to concat
toString() converts back to String

java.lang.Math

```
Math.abs(NUM),Math.ceil(NUM),Math.floor(NUM)
,Math.log(NUM),Math.max(A,B),Math.min(C,D),
Math.pow(A,B),Math.round(A),Math.random()
```

IF STATEMENTS:

```
if( boolean_value ) { STATEMENTS }
else if( bool )     { STATEMENTS }
else if( ..etc )    { STATEMENTS }
else                { STATEMENTS }
//curly brackets optional if one line
```

LOOPS:

```
while( bool )         { STATEMENTS }
for(INIT;BOOL;UPDATE) { STATEMENTS }
//1INIT 2BOOL 3STATEMENTS 4UPDATE 5->Step2
do{ STATEMENTS }while( bool );
//do loops run at least once before checking
break;    //ends enclosing loop (exit loop)
continue; //jumps to bottom of loop
```

ARRAYS:

```
int[] x = new int[10]; //ten zeros
int[][] x = new int[5][5]; //5 by 5 matrix
int[] x = {1,2,3,4};
x.length; //int expression length of array
int[][] x = {{1,2},{3,4,5}}; //ragged array
String[] y = new String[10]; //10 nulls
//Note that object types are null by default

//loop through array:
for(int i=0;i<arrayname.length;i++) {
  //use arrayname[i];
}

//for-each loop through array
int[] x = {10,20,30,40};
for(int v : x) {
  //v cycles between 10,20,30,40
}

//Loop through ragged arrays:
for(int i=0;i<x.length;i++)
  for(int j=0;j<x[i].length;j++) {
     //CODE HERE
  }

//Note, multi-dim arrays can have nulls
//in many places, especially object arrays:
Integer[][] x = {{1,2},{3,null},null};
```

FUNCTIONS / METHODS:

```
Static Declarations:
public static int functionname( … )
private static double functionname( … )
static void functionname( … )
Instance Declarations:
public void functionname( … )
private int functionname( … )
Arguments, Return Statement:
int myfunc(int arg0, String arg1) {
  return 5; //type matches int myfunc
}
//Non-void methods must return before ending
//Recursive functions should have an if
//statement base-case that returns at once
```

CLASS/OBJECT TYPES:

```
INSTANTIATION:
public class Ball {//only 1 public per file
  //STATIC FIELDS/METHODS
  private static int numBalls = 0;
  public static int getNumBalls() {
    return numBalls;
  }
  public static final int BALLRADIUS = 5;

  //INSTANCE FIELDS
  private int x, y, vx, vy;
  public boolean randomPos = false;

  //CONSTRUCTORS
  public Ball(int x, int y, int vx, int vy)
  {
    this.x = x;
    this.y = y;
    this.vx = vx;
    this.vy = vy;
    numBalls++;
  }
  Ball() {
    x = Math.random()*100;
    y = Math.random()*200;
    randomPos = true;
  }

  //INSTANCE METHODS
  public int getX(){ return x; }
  public int getY(){ return y; }
  public int getVX(){ return vx; }
  public int getVY(){ return vy; }
  public void move(){ x+=vx; y+=vy; }
  public boolean touching(Ball other) {
    float dx = x-other.x;
    float dy = y-other.y;
    float rr = BALLRADIUS;
    return Math.sqrt(dx*dx+dy*dy)<rr;
  }

}

//Example Usage:
public static void main(String[] args) {
  Ball x = new Ball(5,10,2,2);
  Ball y = new Ball();
  List<Ball> balls = new ArrayList<Ball>();
  balls.add(x); balls.add(y);
  for(Ball b : balls) {
    for(Ball o : balls) {
      if(b != o) { //compares references
        boolean touch = b.touching(o);
      }
    }
  }
}
```

® 2013 Kapparate, © 2013 Kapparate Operating LLC

POLYMORPHISM:

Single Inheritance with "extends"

```
class A{ }
class B extends A{ }
abstract class C { }
class D extends C { }
class E extends D
```

Abstract methods

```
abstract class F {
  abstract int bla();
}
class G extends F {
  int bla() {  //required method
   return 5;
  }
}
```

Multiple Inheritance of interfaces with "implements" (fields not inherited)

```
interface H {
  void methodA();
  boolean methodB(int arg);
}
interface I extends H{
  void methodC();
}
interface K {}
class J extends F implements I, K {
  int bla() { return 5; } //required from F
  void methodA(){} //required from H
  boolean methodB(int a) { //req from A
    return 1;
  }
  void methodC(){} //required from I
}
```

Type inference:

```
A x = new B(); //OK
B y = new A(); //Not OK
C z = new C(); //Cannot instantiate abstract
//Method calls care about right hand type
(the instantiated object)
//Compiler checks depend on left hand type
```

GENERICS:

```
class MyClass<T> {
  T value;
  T getValue() { return value; }
}
class ExampleTwo<A,B> {
  A x;
  B y;
}
class ExampleThree<A extends List<B>,B> {
  A list;
  B head;
}
//Note the extends keyword here applies as
well to interfaces, so A can be an interface
that extends List<B>
```

JAVA COLLECTIONS:

List<T>: Similar to arrays

```
ArrayList<T>: Slow insert into middle
//ArrayList has fast random access
LinkedList<T>: slow random access
//LinkedList fast as queue/stack
Stack: Removes and adds from end
```

List Usage:

```
boolean add(T e);
void clear(); //empties
boolean contains(Object o);
T get(int index);
T remove(int index);
boolean remove(Object o);
//remove uses comparator
T set(int index, E val);
Int size();
```

List Traversal:

```
for(int i=0i<x.size();i++) {
       //use x.get(i);
}

//Assuming List<T>:
for(T e : x) {
       //use e
}
```

Queue<T>: Remove end, Insert beginning

```
LinkedList implements Queue
```

Queue Usage:

```
T element(); // does not remove
boolean offer(T o); //adds
T peek(); //pike element
T poll(); //removes
T remove(); //like poll
Traversal: for(T e : x) {}
```

Set<T>: uses Comparable<T> for uniqueness

```
TreeSet<T>, items are sorted
HashSet<T>, not sorted, no order
LinkedHashSet<T>, ordered by insert
Usage like list: add, remove, size
Traversal: for(T e : x) {}
```

Map<K,V>: Pairs where keys are unique

```
HashMap<K,V>, no order
LinkedHashMap<K,V> ordered by insert
TreeMap<K,V> sorted by keys

V get(K key);
Set<K> keySet(); //set of keys
V put(K key, V value);
V remove(K key);
Int size();
Collection<V> values(); //all values

Traversal: for-each w/ keyset/values
```

java.util.PriorityQueue<T>

A queue that is always automatically sorted using the comparable function of an object

```
public static void main(String[] args) {
  Comparator<String> cmp= new LenCmp();
  PriorityQueue<String> queue =
     new PriorityQueue<String>(10, cmp);
  queue.add("short");
  queue.add("very long indeed");
  queue.add("medium");
  while (queue.size() != 0)
    System.out.println(queue.remove());
}
class LenCmp implements Comparator<String> {
  public int compare(String x, String y){
    return x.length() - y.length();
  }
}
```

java.util.Collections algorithms

Sort Example:

```
//Assuming List<T> x
Collections.sort(x); //sorts with comparator
```

Sort Using Comparator:

```
Collections.sort(x, new Comparator<T>{
  public int compareTo(T a, T b) {
    //calculate which is first
    //return -1, 0, or 1 for order:
    return someint;
  }
}
```

Example of two dimensional array sort:

```
public static void main(final String[] a){
   final String[][] data = new String[][] {
     new String[] { "20090725", "A" },
     new String[] { "20090726", "B" },
     new String[] { "20090727", "C" },
     new String[] { "20090728", "D" } };
     Arrays.sort(data,
       new Comparator<String[]>() {
        public int compare(final String[]
entry1, final String[] entry2) {
          final String time1 = entry1[0];
          final String time2 = entry2[0];
          return time1.compareTo(time2);
        }
     });

     for (final String[] s : data) {
       System.out.println(s[0]+" "+s[1]);
     }
   }
}
```

More collections static methods:

```
Collections.max( … ); //returns maximum
Collections.min( … ); //returns maximum
Collections.copy( A, B); //A list into B
Collections.reverse( A ); //if A is list
```

® 2013 Kapparate, © 2013 Kapparate Operating LLC

Maven2 Reference

Invoking Maven

General Syntax:

```
mvn plugin:target [-Doption1 -Doption2 dots]

mvn help
mvn -X ...
```

Prints help debugging output, very useful to diagnose

Creating a new Project (jar)

```
mvn archetype:create -DgroupId=Artifact Group

                     -DartifactId=Artifact ID
```

Example:

```
mvn archetype:create -DgroupId=de.focusdv.bcs

                     -DartifactId=new-app
```

Creates a new Project Directory *new-app* with package structure *de.focusdv.bcs*.

Name of the packaged jar will be new-app-*version*.jar

Creating a new Project (war)

```
mvn archetype:create
        -DgroupId=Artifact Group
        -DartifactId=Artifact ID
        -DarchetypeArtifactId=maven-archetype-webapp
```

Example:

```
mvn archetype:create
        -DgroupId=de.focusdv.bcs
        -DartifactId=new-webapp
        -DarchetypeArtifactId=maven-archetype-webapp
```

Creates a new Directory *new-webapp* with package structure *de.focusdv.bcs*.

Name of the packaged war will be new-app-*version*.war

Standard Project Structure

directory	description
/new-app/pom.xml	maven2 project file
/new-app/src/	Sources
/new-app/src/main/java/	Java source tree
/new-app/src/test/java/	Java unit tests
/new-app/src/main/resources/	Java classpath resources
/new-app/src/test/resources/	Resources for unit-tests
/new-app/target/classes/	compiles classes
/new-app/target/test-classes/	compiles test classes
/new-app/target/dots	other plugins' output

/new-webapp/src/main/webapp	root of webapp

Compiling

```
mvn compile
```

Running Unit Tests / Code Coverage

```
mvn test
```

compiles and runs unit tests

```
mvn clean cobertura:cobertura
```

generates a code-coverage report for the tests. It only works, if the pom.xml is configured as follows:

```
</project>
  …
  <build>
    <plugins>
     …
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
        <executions>
          <execution>
            <goals>
              <goal>clean</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
     …
    </plugins>
  </build>
  …
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </reporting>
  …
</project>
```

Packaging (jar, war)

```
mvn clean package
```

compiles, runs unit tests and packages the artifact (clean makes sure there are no unwanted files in the package)

Installing Artifact in Local Repository

```
mvn clean install
```

compiles, runs unit tests, packages and installs the artifact in the local repository. (*User Home Directory*/.m2/repository/)

Installing 3rdParty jar in local Repository

```
mvn install:install-file -Dfile=foo.jar

    -DgroupId=org.foosoft -DartifactId=foo

    -Dversion=1.2.3 -Dpackaging=jar
```

Cleaning Up

```
mvn clean
```

Creating Eclipse Project Structure

```
mvn eclipse:eclipse
```

If using the eclipse plugin from update-site

http://m2eclipse.codehaus.org

remove the generated dependencies from project.

Maven Project file (pom.xml)

Minimal pom.xml is created with

```
mvn archetype:create
```

(see above).

Adding Dependencies

```
<project>
  …
 <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring</artifactId>
      <version>1.2.6</version>
    </dependency>
    …
  </dependencies>
```

Because of , junit will not be included in final packaging.

Adding Developers

```
<project>
  …
  <developers>
    <developer>
```

```
      <id>Baier</id>
      <name>Hans Baier</name>
      <email>hans.baier::at::focus-dv.de</email>
      <organization>focus DV GmbH</organization>
      <roles>
        <role>Developer</role>
      </roles>
    </developer>
    …
  </developers>
```

Setting Compiler Version

```
<project>
 …
 <build>
  <plugins>
   <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
      <source>1.5</source>
      <target>1.5</target>
    </configuration>
   </plugin>
   …
  </plugins>
 </build>
```

Assemblies and Profiles

Creating Assemblies

To package the artifact use the following lines in the .pom-file:

```
<plugin>
   <artifactId>maven-assembly-plugin</artifactId>
   <configuration>
     <descriptors>
<descriptor>src/main/assembly/foo-dep.xml</descriptor>
<descriptor>src/main/assembly/foo.xml</descriptor>
     </descriptors>
   </configuration>
 </plugin>
```

src/main/assembly is the maven standard directory for assemblies.

The first assembly descriptor packages all dependencies into one jar:

```
<assembly>
  <id>dep</id>
  <formats>
    <format>jar</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <dependencySets>
    <dependencySet>
      <outputDirectory></outputDirectory>
      <unpack>true</unpack>
      <scope>runtime</scope>
      <excludes>
```

```
        <exclude>junit:junit</exclude>
      </excludes>
    </dependencySet>
  </dependencySets>
</assembly>
```

The second descriptor packages the program:

```
<assembly>
  <id>bin</id>
  <formats>
    <format>zip</format>
  </formats>
  <fileSets>
    <fileSet>
      <directory>src/main/assembly/files</directory>
      <outputDirectory></outputDirectory>
      <includes>
        <include>**/*.bat</include>
        <include>**/native/**</include>
        <include>**/*.properties</include>
      </includes>
    </fileSet>
    <fileSet>
      <directory>target</directory>
      <outputDirectory></outputDirectory>
      <includes>
        <include>*.jar</include>
      </includes>
    </fileSet>
  </fileSets>
</assembly>
```

Supplementary files in this example are in

src/main/assembly/files.

This includes the program starter (.bat), native libraries (/native) and Properties files.

Packaging is invoked by:

```
mvn assembly:assembly
```

Using Profiles

Profiles enable different versions of a project to be build, or adapting to different environments by an option on the command line. Profiles can modify almost all dependencies, plugins and settings in the *pom.xml*. In *cockpit-model* they are used to generate a restricted demo-version and a release-version like that:

```
<profiles>
    <profile>
      <id>release-profile</id>
      <dependencies>
        <dependency>
          <groupId>swt</groupId>
          <artifactId>swt-win32</artifactId>
          <version>3.2.1</version>
        </dependency>
      </dependencies>
      <build>
        <filters>
          <filter>src/main/filters/releaseVersion.properties</filter>
```

```
        </filters>
      </build>
    </profile>
    <profile>
      <id>demo</id>
      <dependencies>
        <dependency>
          <groupId>swt</groupId>
          <artifactId>swt-win32</artifactId>
          <version>3.2.1</version>
        </dependency>
      </dependencies>
      <build>
        <filters>
          <filter>src/main/filters/demoVersion.properties</filter>
        </filters>
      </build>
    </profile>
    …
  </profiles>
```

Here the *release-profile* uses the windows library of SWT (since our customers' platform is windows (like it or not...), and substitutes the resources files' placeholders with the variables in *releaseVersion.properties*. The *demo*-profile is almost the same except it uses *demoVersion.properties* for filtering.

Usage:

```
mvn -Prelease-profile clean assembly:assembly
```

or

```
mvn -Pdemo clean assembly:assembly
```

Using Profiles by OS

In this example we want to use the Linux SWT Libraries on Linux and the Windows libs on Windows:

```
<profiles>
  <profile>
   <id>windows</id>
   <activation>
    <os>
     <family>windows</family>
    </os>
   </activation>
   <dependencies>
    <dependency>
     <groupId>swt</groupId>
     <artifactId>swt-win32</artifactId>
     <version>3.1.1</version>
    </dependency>
   </dependencies>
  </profile>
  <profile>
   <id>unix</id>
   <activation>
    <os>
     <family>unix</family>
    </os>
   </activation>
   <dependencies>
```

```
    <dependency>
     <groupId>swt</groupId>
     <artifactId>swt-linux-gtk</artifactId>
     <version>3.1.1</version>
    </dependency>
   </dependencies>
  </profile>
 </profiles>
```

Versioning, Repositories and Releases

Setting Source Code Control System

```
<project>
 …
 <scm>
   <developerConnection>
scm:svn:https://svnhost.net/svnroot/trunk/new-app
   </developerConnection>
 </scm>
 <build>
  <plugins>
   <plugin>
    <artifactId>maven-release-plugin</artifactId>
    <configuration>
     <tagBase>
       https://svnhost.net/svnroot/tags
     </tagBase>
    </configuration>
   </plugin>
   …
  </plugins>
 </build>
```

Versioning

Keep the Verision of your POM artifact in the form *version*-SNAPSHOT until you release.

Mavens release plugin then removes the -SNAPSHOT suffix.

Using internal Repositories

This assumes that a machine *myhost* exists with a configured and running Web-Server and SSH-Server

```
<repositories>
 <repository>
  <id>focus-repository</id>
  <name>Focus BCS Repository</name>
  <url>http://myhost/mvn/repository</url>
 </repository>
</repositories>
<distributionManagement>
 <repository>
  <id>focus-repository</id>
  <name>Focus BCS Repository</name>
  <url>scp://myhost/var/www/mvn/repository/</url>
 </repository>
</distributionManagement>
```

Installing Artifact in Remote Repository

```
mvn clean deploy
```

compiles, runs unit tests, packages and installs the artifact in the remote repository.

Install 3rdParty jar to Remote Repository

```
mvn deploy:deploy-file -DgroupId=commons-collections

  -DartifactId=collections-generic -Dversion=4.0

  -Dpackaging=jar -Dfile=collections-generic-4.0.jar

  -DrepositoryId=focus-repository

  -Durl=scp://host/home/mvn/public_html/repository
```

Preparing Releases

Make sure, the SCM settings in the POM are correct and all changes are committed to the SCM. Then execute

```
mvn -Dusername=USER -Dpassword=PASS release:prepare
```

Before issuing the above command use it with *-DdryRun=true* first

tags in configured build profiles in the pom.xml

Performing Releases

```
mvn -P profile -Drelease:perform
```

Checks out the released version from tag in repository, builds, tests, packages and installs package, javadoc and sources in repository. As preparing the release removes activation tags from build profiles, it is necessary to supply the profile or the release will fail.

Web-Development

Integration-Test with tomcat

```
<project>
 …
 <build>
  <plugins>
    …
    <plugin>
      <groupId>org.codehaus.cargo</groupId>
      <artifactId>cargo-maven2-plugin</artifactId>
      <executions>
        <execution>
          <id>tomcat-execution</id>
          <phase>package</phase>
          <goals>
            <goal>start</goal>
          </goals>
          <configuration>
            <wait>true</wait>
```

```
            <container>
             <containerId>tomcat5x</containerId>
             <zipUrlInstaller>
<url><http://www.apache.org/.../jakarta-tomcat.zip></url>
              <installDir>${installDir}</installDir>
             </zipUrlInstaller>
            </container>
            <configuration>
<dir>${project.build.directory}/tomcat5x/</dir>
            </configuration>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
 </build>
```

Then execute in project directory:

```
mvn -X integration-test
```

The war-file will built, tested and packaged. Then tomcat will be downloaded, installed and started with the war-file of the project deployed to the server.

If you want to use jetty4 (already embedded, fast startup) use:

```
mvn cargo:start
```

(Press Ctrl-C to stop)

Online web-development with Jetty plugin

Add Maven-Plugin to pom.xml:

```
<plugins>
     …
      <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>maven-jetty6-plugin</artifactId>
        <configuration>
          <scanIntervalSeconds>10</scanIntervalSeconds>
        </configuration>
      </plugin>
   …
   </plugins>
```

Then run Jetty with

```
mvn jetty6:run
```

Online web-development and automatic deployment with tomcat plugin

Add Maven-Plugin to pom.xml:

```
<plugins>
     …
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>tomcat-maven-plugin</artifactId>
        <configuration>
          <url>http://192.168.129.36:8080/manager/html</url>
        </configuration>
      </plugin>
```

```
      <plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven2-plugin</artifactId>
      </plugin>
     …
   </plugins>
…
  <repositories>
    <repository>
      <id>codehaus</id>
      <name>Codehaus maven repository</name>
      <url>http://dist.codehaus.org/</url>
      <layout>legacy</layout>
    </repository>
    …
  </repositories>
```

Then run Tomcat with

```
mvn tomcat:run
```

Deploy the war automatically with

```
mvn tomcat:deploy
```

If already deployed, the webapp needs to be undeployed first:

```
mvn tomcat:undeploy
```

Note that automatic deployment/undeployment only works without further configuration in *$MAVEN2_HOME/conf/settings.xml* if the managers username is admin with empty password