Java reference/summary

Because coding can be challenging enough without trying to memorise everything!

This guide is intended as a helpful reminder to some of the quirks of Java syntax and it's various libraries you need to use. Please suggest changes/improvements as you spot things. Thanks!

Numeric variables and operations

Standard datatypes

```
byte b = 1;          // 8 bits
short s = 2;         // 16 bits
int a = 1;           // 32 bits
long l = 1;          // 64 bits
float f = 1.0;       // 32 bits
double d = 1.0;      // 64 btts
boolean b = false;
int hex = 0xFF;      // hex FF = decimal 255
```

Numerical operations

```
int a = 5;
int b = 4;

int answer = a + b;                 // addition
int answer = a - b;                         // subtraction
int answer = a * b;                         // Multiplication
int answer = a / b;                         // Integer division
int answer = a % b;                 // Modulus (remainder)

double answer = Math.pow(a,b);      // Exponential
double answer = Math.sqrt(a);       // Square root

int answer = Math.round( 13.4 );    // Rounding
int answer = Math.abs( -13 );       // Absolute value
float random = Math.random();       // Between 0 & 1
```

Integer v Float devision

Determined automatically based on the datatypes.

```
int a = 20;
int b = 6;
System.out.println( a / b );
System.out.println( a % b );

double c = 20;
double d = 6;
System.out.println( c / d );
System.out.println( c % d );
```

PEMDAS

```
System.out.println( 2 + 4 * 3 );                // 14 or 18?
```

Trigonometry

```
// All angles will be in radians not degrees
double pi = Math.PI;
double angle = Math.sin( opp / hyp );
double angle = Math.cos( adj / hyp );
double angle = Math.tan( opp / adj );
double ratio = Math.asin( angle );
double ratio = Math.acos( angle );
double ratio = Math.atan( angle );
```

System input and output

Printing

```
System.out.print()                  // Without adding a new line at the end
System.out.println()                // With a new line at the end

System.out.println("Hello world"!);

String message = "Hello world";
System.out.println( message );
```

Input

String input

```
java.util.Scanner reader = new java.util.Scanner(System.in);
String t = reader.nextLine();
```

Numeric input

```
java.util.Scanner reader = new java.util.Scanner(System.in);

System.out.print("Type a number:");
int i = reader.nextInt();

System.out.print("You typed ");
System.out.print( i );
System.out.print(", the next number is ");
System.out.println( i+1 );
```

Strings and operations

String functions

```
String s1 = "hello";
String s2 = "What does the fox say?";

s1.length();                    // 5
s1.charAt(0);                   // "h"
s1.codePointAt(0);              // 104 (unicode)
s2.indexOf("fox");              // 14
s2.lastIndexOf("fox");          // 14
s2.substring(14,17);            // fox
s2.replace("fox","goat");       // What does the goat say?
s2.toUpperCase();                     // WHAT DOES THE FOX SAY?
s2.toLowerCase();                     // what does the fox say?

String s3 = "Hi there! " + s2;  // concatenation

String[] words = s2.split(" ");  // ["What","does","the","fox","say?"]
```

Want to change an individual letter inside a string? Unlike other languages, you can not do `str[2] = 'x'` or similar. Java Strings are immutable (unchangeable). Most "simple" solution...

```
String myName = "halftime";
myName = myName.substring(0,4) + 'x' + myName.substring(5);
System.out.println(myName);                     // halfxime
```

Casting between primitive datatypes

Casting means converting between datatypes

```
// Given these variables
byte b = 1;
short s = 2;
int i = 3;
long l = 4;
float f = 5.0F;
double d = 6.0;
String s1 = "4";
String s2 = "5.3";
char c = '7';

// To integers
int num1 = Integer.parseInt( s1 );
int num2 = Integer.parseInt( s2 );          // RUN TIME ERROR
int num3 = (int)f;
int num4 = (int)d;
int num5 = (int)l;
int num6 = (int)c;

// To long
long l1 = Long.parseLong( s1 );
long l2 = Long.parseLong( s2 );             // RUN TIME ERROR
long l3 = (long)f;
long l4 = (long)d;
long l5 = (long)l;
long l6 = (long)c;

// To float
float fl1 = Float.parseFloat( s1 );
float fl2 = Float.parseFloat( s2 );
float fl3 = (float)i;
float fl4 = (float)l;
float fl5 = (float)d;
float fl6 = (float)c;

// To doubles
double do1 = Double.parseDouble( s1 );
double do2 = Double.parseDouble( s2 );
double do3 = (double)i;
double do4 = (double)l;
double do5 = (double)d;
double do6 = (double)c;

// To strings
String str1 = Integer.toString( i );
String str2 = Float.toString( f );
String str3 = Double.toString( d );
String str4 = Long.toString( l );
String str5 = Character.toString( c );
```

Conditional execution

Numeric comparisons

```
(a == b)                                IS EQUAL TO
(a != b)                                IS NOT EQUAL TO
(a < b)                           IS LESS THAN
(a <= b)                                IS LESS THAN OR EQUAL TO
(a > b)                           IS GREATER THAN
(a >= b)                                IS GREATER THAN OR EQUAL TO
```

String comparisons

Equality check

```
Objects.equals(s1, s2)          // USE THIS
s1.equals(s2)                           // OLD METHOD YOU'LL SEE USED ONLINE
```

Comparing order

```
s1.compareTo(s2)
```

== 0 when string values match
== negative when s1 is alphabetically 1st
== positive when s1 is alphabetically 2nd

.

Multiple comparisons

```
( condition1 && condition2 )            // AND
( condition1 || condition2 )            // OR
(! condition1 )                                 // NOT
```

If construct

```
if ( condition ) {
   doSomething;
   doSomething;
} else if ( condition ) {
   doSomething;
   doSomething;
} else {
   doSomething;
   doSomething;
}
```

For construct

```
for ( initialization ; comparison ; iteration ) {
   instructions();
}
```

Example:

```
for (int i=0 ; i<10 ; i=i+1 ) {
   System.out.println( i );
}
```

While construct

```
while ( comparison ) {
   instructions();
}
```

Example:

```
int a = 0;
while ( a < 10 ) {
   a = a + 1;
   System.out.println( a );
}
```

Ternary operator

Also known as the "if statement in one line"

Syntax:

```
boolean result = (condition) ? result_if_true : result_if_false ;
```

Example:

```
int largerOfTheTwo = (a>b) ? a : b;
```

Static 1 dimensional arrays

Declaring static arrays: Method 1

```
int[] primes = new int[10];
primes[0] = 1;
primes[1] = 2;
primes[2] = 3;
primes[3] = 5;
primes[4] = 7;
primes[5] = 11;
primes[6] = 13;
primes[7] = 17;
primes[8] = 19;
primes[9] = 23;
```

Declaring static arrays: Method 2

```
int[] primes = {1,2,3,5,7,11,13,17,19,23};
```

For loop thorugh an array

```
for (int item : primes) {
   System.out.println( item );
}

for (int i=0; i<primes.length; i++) {
   System.out.println( primes[i] );
}
```

Static array comparisons

```
import java.util.Arrays;

// Check if two arrays are filled with matching values
if ( Arrays.equals( primes, other )) {
    System.out.println("The two arrays match");
}

// Length of an array
int l = primes.length;

// Sort an array in ascending order
Arrays.sort( primes );

// Create a string listing the contents of the array
// output: [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]
System.out.println( Arrays.toString( primes ));
```

Array Lists

Import

```
import java.util.ArrayList;
```

ArrayList: Declaring

Note that when instantiating an ArrayList, yyou can nominate the datatype to be contained by the array list or leave it unspecified (to allow for a potential mix of different data types). There is an example of each below.

Example 1: Create an ArrayList specifying elements to be String

```
ArrayList<String> alist = new ArrayList<String>();
alist.add("first");
alist.add("second");

for (String item: list) {
    System.out.println( item );
}
```

Example 2: Create an ArrayList without specifying element type

```
ArrayList list = new ArrayList();
list.add( new Person("John Doe") );
list.add( new Person("Jane Smith") );
list.add( "Mixed data arrayList" );
list.add( 99 );

for (Object o : list) {
    if (o.getClass() == Person.class) {
        System.out.println("The person was "+o.toString());
    } else if (o.getClass() == String.class) {
        System.out.println("The string was "+o);
    } else if (o.getClass() == Integer.class) {
        System.out.println("The integer was "+o);
    }
}
```

ArrayList methods

- list.add(o) - Add object o to the end of the list
- list.add(i, o); - Add object o at position index i
- list.get(i) - Returns the object at position index i
- list.push(o) - Add object o to the end of the list
- list.pop() - Returns the last item on the list and removes it
- list.remove(i); - Remove object at index position i
- list.size() - The number of items in the list

ArrayList: Converting between static arrays

Array to ArrayList

```
ArrayList arrlist = new ArrayList<>(Arrays.asList(arr))
```

ArrayList to Array
(assuming it is String... change as appropriate)

```
String[] arr = (String[])arrlist.toArray(new String[arrlist.size()]);
```

Example: Load file to ArrayList

One line per string

```
include java.io.File;
include java.utils.ArrayList;
include java.utils.Scanner;
...
File f = new File("/path/to/filename.ext");
Scanner reader = new Scanner(file);
ArrayList<String> content = new ArrayList<String>();
while (reader.hasNextLine()) {
   content.add(reader.nextLine());
}
reader.close();
```

Static 2 dimensional arrays

Two dimensional array

```
int [][] a = {
   { 1,  2,  3,  4,  5 },
   { 11, 12, 13, 14, 15 },
   { 21, 22, 23, 24, 25 }
};
System.out.println( a[i][j] );
```

Two dimensional array where values are not pre-known

```
int [][] a = new int[3][5];
```

To iterate over the 2D array

```
for (int[] row : a ) {
    for (int cell : row) {
        System.out.println( cell );
    }
}
```

Hash Maps

```
import java.util.HashMap;
import java.util.Iterator;

HashMap people = new HashMap();

Person p1 = new Person("P","Baumgarten",21,3);
people.put("Mr B", p1);

people.put("Mr B", new Person("P","Baumgarten",21,3));
people.put("Alex", new Person("Alex","Buchler",16,7));
people.put("Lennard", new Person("Lennard","Ruess",17,42));
people.put("Fred", new Person("Fred","Falk",17,0));
people.put("Conrad", new Person("Conrad","G-B",17,-5));

for (Object o : people.values()) {
    System.out.println( ((Person)o).toString() );
}

Person p = (Person)people.get("Mr B");
System.out.println( p.toString() );
```

Linked Lists

```
import java.util.LinkedList;

LinkedList s = new LinkedList();
s.push("person 1");
s.push("person 2");
s.push("person 3");
s.push("person 4");
s.pop();
s.push("person 5");
s.push("person 6");
while (! s.isEmpty() ){
    System.out.println( s.pop() );
}
```

.
.
.
.

Functions

Syntax:

```
public static returnType functionName(paramType param1, … ) {
   // insert code
   return value;
}
```

Example: Load file

```
public static String[] getFileAsArray( String filename ) {
   try {
      File f = new File(filename);
            Scanner reader = new Scanner(f);
            ArrayList<String> content = new ArrayList<String>();
            while (reader.hasNextLine()) {
               content.add(reader.nextLine());
            }
            reader.close();
            return((String[])content.toArray(new String[content.size()]));
        } catch (Exception e) {
           System.out.println("ERROR processing file "+filename);
             System.out.println(e.getMessage());
             return(new String[0]);
        }
}

public static void main(String[] args) {
   String[] words = getFileAsArray("dictionary.txt");
     for (String word : words) {
        System.out.println(word);
        }
        System.out.println("There were "+words.length+" words!");
}
```

Classes & objects

General structure of a Java class

```
package com.example.project;

import java.util.*; // or whatever

public class MyProject{
    private int instanceVar;

    MyProject() { // Constructor
        instanceVar = 0;
    }

    public void setInstanceVariable( int i ) {
        instanceVar = i;
    }

    public void setInstanceVariableUsingThis( int instanceVar ) {
        this.instanceVar = instanceVar;
    }

    public int getInstanceVariable() {
        return( instanceVar );
    }

    public void static main( String args[] ) {
        MyProject m = new MyProject();
        m.setInstanceVariable( 10 );
        int val = m.getInstanceVariable();
    }

}
```

Inheritance

The magic happens with the use key word **extends**

```
public class Automobile {
    String registration;
    String owner;

    Automobile(String registration, String owner){
        this.registration = registration;
        this.owner = owner;
        System.out.println("Vehicle "+registration+" registered to "+owner);

    }
    String getOwner() {
        return owner;
    }
}

public class Motorcycle extends Automobile {
    String make;
    String model;
    String serialnumber;
    int enginesize;

    Motorcycle(String registration, String owner, String make, String model){
        super(registration, owner);
        this.make = make;
        this.model = model;
    }

    void printRegistration() {
        System.out.println("Motorcycle: "+registration);
        System.out.println("Owner: "+owner);
        System.out.println("Make: "+make);
        System.out.println("Model: "+model);
    }
}

public class Demo {
    public static void main() {
        Motorcycle m = new Motorcycle("VD-12345","John Doe","Harley","Breakout");
        m.printRegistration();
        System.out.println("The owner is: " + m.getOwner() );

    }
}
```

Encapsulation

Access modifiers:

- public – visible to the world
- protected – visible to the package and all sub classes
- package/default/no access modifier provided - visible to the package
- private – visible to the class only

Example:

```
public class A {
   private int x;
   void setX( int x ) {
      this.x = x;
   }
   int getX() {
      return x;
   }
}

public class B {
   public static void main() {
      A a = new A();
      a.setX( 13 );
      System.out.println( a.getX() );
      a.x = 14;      // this will cause an error
      System.out.println( a.getX() );
   }
}
```

Polymorphism: Overloading

Multiple functions with the same name but different parameter inputs. Java will determine which function to run based on which matches the set of parameters you provide.

```
void driveForward() {
   speed = 60; // Use 60km/hr as a default
}

void driveForward(int speed) {
   this.speed = speed;
}
```

Polymorphism: Overriding

When a function in a child class has the same name as a function in the parent class, the child class' version will take precedence.

Example:

```
public class Person {
```

```
public class Person {
   String name;
   Person(String name) {
      this.name = name;
   }
   String getName() {
      return this.name;
   }
}

public class Royalty extends Person {
   Royalty(String name) {
      super(name);
   }
   String getName() {
         return "Your Royal Highness "+name;
   }
}

public class Demo {
   public static void main() {
        Person commoner = new Person("Elizabeth");
            System.out.println( commoner.getName() );
      Royalty queen = new Royalty("Elizabeth");
      System.out.println( queen.getName() );   // Will run the overriden getName() function
   }
}
```

Date & time

Sooner or later every programmer needs to deal with times and dates. Knowing the appropriate functions for the task can be a mind numbing experience, so it's very handy to heep a reference guide nearby! The following is my attempt. Please suggest improvements.

The following is for Java 8 onwards and comes from https://www.tutorialspoint.com/java8/java8_datetime_api.htm
Prior to Java 8, it is recommended to use a 3rd party class such as http://www.joda.org/joda-time/

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;
import java.time.Month;

// Get the current date and time
LocalDateTime currentTime = LocalDateTime.now();
System.out.println("Current DateTime: " + currentTime);      // Current DateTime: 2014-12-09T11:00:

LocalDate date1 = currentTime.toLocalDate();
System.out.println("date1: " + date1);                       // date1: 2014-12-09

Month month = currentTime.getMonth();
int day = currentTime.getDayOfMonth();
int seconds = currentTime.getSecond();
System.out.println("Month: " + month +" day: " + day +"seconds: " + seconds); // Month: DECEMBER d

LocalDateTime date2 = currentTime.withDayOfMonth(10).withYear(2012);
System.out.println("date2: " + date2);                       // date2: 2012-12-10T11:00:45.457

//12 december 2014
LocalDate date3 = LocalDate.of(2014, Month.DECEMBER, 12);
System.out.println("date3: " + date3);                       // date3: 2014-12-12
```

```
// 22 hour 15 minutes
LocalTime date4 = LocalTime.of(22, 15);
System.out.println("date4: " + date4);                       // date4: 22:15

// parse a string
LocalTime date5 = LocalTime.parse("20:15:30");
System.out.println("date5: " + date5);                       // date5: 20:15:30
```

Time zones

```
import java.time.ZonedDateTime;
import java.time.ZoneId;

// Get the current date and time
ZonedDateTime date1 = ZonedDateTime.parse("2007-12-03T10:15:30+05:30[Asia/Karachi]");
System.out.println("date1: " + date1);                       // date1: 2007-12-03T10:15:30+05:00[As

ZoneId id = ZoneId.of("Europe/Paris");
System.out.println("ZoneId: " + id);                         // ZoneId: Europe/Paris

ZoneId currentZone = ZoneId.systemDefault();
System.out.println("CurrentZone: " + currentZone);           // CurrentZone: Etc/UTC
```

Difference between two dates/times

With Java 8, two specialized classes are introduced to deal with the time differences –

- Period – It deals with date based amount of time.
- Duration – It deals with time based amount of time.

```
import java.time.temporal.ChronoUnit;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.Duration;
import java.time.Period;

//Get the current date
LocalDate date1 = LocalDate.now();
System.out.println("Current date: " + date1);

//add 1 month to the current date
LocalDate date2 = date1.plus(1, ChronoUnit.MONTHS);
System.out.println("Next month: " + date2);
Period period = Period.between(date2, date1);
System.out.println("Period: " + period); // P-1M

LocalTime time1 = LocalTime.now();
Duration twoHours = Duration.ofHours(2);
LocalTime time2 = time1.plus(twoHours);
Duration duration = Duration.between(time1, time2);
System.out.println("Duration: " + duration); // PT2H
```

Unix time / Epoch time

```
// Current UTC unix time in seconds
long unixTime = System.currentTimeMillis() / 1000L;
```

```
// A "new" Java 8 function that does the same thing
// import java.time.Instant
Instant instant = Instant.ofEpochMilli(millis);

// Take an existing LocalDate object and get its epoch time
LocalDate date = ...;
ZoneId zoneId = ZoneId.systemDefault(); // or: ZoneId.of("Europe/Oslo");
long epoch = date.atStartOfDay(zoneId).toEpochSecond();

// Take an existing LocalDateTIme object and get its epoch time
LocalDateTime time = ...;
ZoneId zoneId = ZoneId.systemDefault(); // or: ZoneId.of("Europe/Oslo");
long epoch = time.atZone(zoneId).toEpochSecond();
```

Convert from the old java.util.Date object to the new version

```
import java.time.LocalDateTime;
import java.time.ZonedDateTime;
import java.util.Date;
import java.time.Instant;
import java.time.ZoneId;

//Get the current date
Date currentDate = new Date();
System.out.println("Current date: " + currentDate);

//Get the instant of current date in terms of milliseconds
Instant now = currentDate.toInstant();
ZoneId currentZone = ZoneId.systemDefault();

LocalDateTime localDateTime = LocalDateTime.ofInstant(now, currentZone);
System.out.println("Local date: " + localDateTime);

ZonedDateTime zonedDateTime = ZonedDateTime.ofInstant(now, currentZone);
System.out.println("Zoned date: " + zonedDateTime);
```

Unicode

Unicode characters are a quick and easy way to use glyphs, emoji and other symbols in your app without having to create them yourself. Once you know the symbol codes it's just a case of using this code:

The French flag would use the codes: \u1F1EB + \u1F1F7

```
StringBuffer sb = new StringBuffer();
sb.append(Character.toChars(0x1F1EB));
sb.append(Character.toChars(0x1F1F7));
System.out.println(sb);
```

or

```
int[] codepoints = { 0x1F1EB, 0x1F1F7 };
String s = new String(codepoints, 0, codepoints.length);
```

To find the required code, or to browse the available list, visit http://emojipedia.org/ and scroll to the bottom of the page for any emoji to find it's "codepoint".

There are also other "non-emoji" symbols that could come in useful, so search sites such as:
https://unicode-table.com/en/#miscellaneous-symbols

JSON

An incomplete guide to using JSON with Java

From http://www.oracle.com/technetwork/articles/java/json-1973242.html

Import

```
import javax.json.*;
```

JSON (Javascript Object Notation) is a commonly used form of structured text to transfer data over networks.

```
{
    "data" : [
        { "from" : { "name" : "xxx", ... }, "message" : "yyy", ... },
        { "from" : { "name" : "ppp", ... }, "message" : "qqq", ... },
        ...
    ],
    ...
}
```

```
URL url = new URL("https://graph.facebook.com/search?q=java&type=post");
try (InputStream is = url.openStream();
    JsonReader rdr = Json.createReader(is)) {
    JsonObject obj = rdr.readObject();
    JsonArray results = obj.getJsonArray("data");

    for (JsonObject result : results.getValuesAs(JsonObject.class)) {
        System.out.print(result.getJsonObject("from").getString("name"));
        System.out.print(": ");
        System.out.println(result.getString("message", ""));
        System.out.println("-----------");
    }
}
```

Class	Role
Json	Contains static methods to create JSON readers, writers, builders, and their factory objects.
JsonGenerator	Writes JSON data to a stream one value at a time.
JsonReader	Reads JSON data from a stream and creates an object model in memory.
JsonObjectBuilder	Create an object model in memory by adding values from application code.
JsonArrayBuilder	Create an array model in memory by adding values from application code.
JsonWriter	Writes an object model from memory to a stream.
JsonValue	

Class	Role
JsonObject	
JsonArray	
JsonString	
JsonNumber	Represent data types for values in JSON data.

Building JSON objects

For example 1: An empty JSON object can be built as follows:

```
JsonObject object = Json.createObjectBuilder().build();
```

For example 2: The following JSON

```
{
    "firstName": "John", "lastName": "Smith", "age": 25,
    "address" : {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

can be built using :

```
JsonObject value = Json.createObjectBuilder()
    .add("firstName", "John")
    .add("lastName", "Smith")
    .add("age", 25)
    .add("address", Json.createObjectBuilder()
        .add("streetAddress", "21 2nd Street")
        .add("city", "New York")
        .add("state", "NY")
        .add("postalCode", "10021"))
    .add("phoneNumber", Json.createArrayBuilder()
        .add(Json.createObjectBuilder()
            .add("type", "home")
            .add("number", "212 555-1234"))
        .add(Json.createObjectBuilder()
            .add("type", "fax")
            .add("number", "646 555-4567")))
    .build();
```

Retrieving JSON values

```
String firstName = jsonobject.getString("firstName");
```