Java – 1/26/2019

8 Primitive types: byte, short, int, long, float, double, char (16-bit Unicode), boolean; String  
int -2\_147\_483\_648 to 2\_147\_483\_647 , -2.1e9 to 2.1e9   
double 4.9e-324 to 1.8e+308, (64 bits: 1 sign, 11 exponent, 52 number; 15-17 digit precision)  
long -9\_223\_372\_036\_854\_775\_805 to 9\_223\_372\_036\_854\_775\_807 , -9.2e18 to 9.2e18  
byte, short, and char are automatically promoted to int in expressions; decimal literals are double   
Casting big to small type: (int)5.55, int i=129; byte b=(byte)i; // b=-127; float a=1.2f  
Type Wrappers: Integer, Long, Double, Character, Boolean, Byte, Short, Float   
Conversions: s=Integer.toString(i); s=i+””; int i = Integer.parseInt(st); .valueOf(st);  
Integer: MAX\_VALUE, MIN\_VALUE, bitCount(), parseInt(st), valueOf(st), toHexString(), toString(i)  
Character: .isUpperCase(c), .toLowerCase(c), .isLetter(c), .isDigit(c) // char 0-65535  
Int Literals: octal (leading 0), hex (leading 0x 0X), binary (leading 0b 0B), long (append L), \_  
int oct = 010, hex = 0x10, bin = 0b10, sum = oct+hex+bin; // outputs are decimal 8,16,2, sum=26  
Integer.toOctalString(sum) .toHexString(sum) .toBinaryString(sum) // 32, 1a, 11010  
Unicode char: \uxxxx (hex), \n – new line, \”, \’ st = Long.toHexString(lnum);  
final double PI = 3.14, hexPi = 0x1.8p1; // p is hex floating point num, 1.5 dec x 2^1 = 3.0

Operators: + - / \* %, ++, --, +=, -=, \*=, /=, %=, a=1;b=a++; c=1;d=++c; //b=1, d=2  
long a=10; int b=4; b=b\*a; //does not compile; b\*=a; // short for b=(int)(b\*a);  
Relational: ==, !=, >=, <=, >, <;   
Boolean Logical Operators: &&, ||, !, ?: int a = (b>c) ? 3 : 1; Short-circuit: && and ||   
Bit operators: &, |, ^, ~, <<, >> (-8 >> 1 =-4), >>> (zero fill); Two’s complement for negative #s

for(int i=-129;i<129;i++){Integer a=i;Integer b=i;Boolean c= a==b;} //-128<=i<=127, c=true; cached  
Object references compare the System.identityHashCode(o) in ==. Use ob1.equals(ob2).

Math: Math.max, min, abs, pow(a,b), sqrt [is accurate], ceil, floor, round, signum, random, sin, cos, log10, log (base e); Comments: //line, /\* spans line(s) \*/ /\*\* documentation \*/

Process-oriented model vs. object-oriented programming;   
Manage complexity through Abstraction; 3 OOP Principles: Encapsulation, Inheritance, Polymorphism.  
class Hello { public static void main(String args[]) {System.out.println(“Hello World”); } }  
System.out.println(), .print(), System.in.read (read from keyboard) append throws IOException  
Compile: c:\>javac Hello.java (compiles to Hello.class) Run: c:\>java Hello (runs Hello.class)

if (x>0 || y==0 && z<1) {} else {}  
for (int i=0; i < 10; i++) {}, while (x < 10) {}, do {} while (x < 10);  
outer: for… continue/break outer; for (char c : st.toCharArray()) {}  
switch (val) {case 2: case3: action; break; case 30: action; break; default: action; break;}

String s = ”d”; char ch[]={‘a’,’b’,’c’}; String s = new String(chars); String(chars,1,2);  
.length(), .charAt(), .toCharArray(), .equals(st2), .equalsIgnoreCase(), .compareTo(): <0, >0, 0.   
.startsWith(), .endsWith(), .indexOf(), .lastIndexOf(), .contains();  
s = s.substring(stInd, endInd+1), .trim(), .toUpperCase(), .toLowerCase(), .replace(‘a’, ‘b’);  
String s = String.valueOf(i); s=””+i;  
String[] st = s.split(“ “); s.split(“[+-]”); s.split(“(\s+)”); s.split(“[.]”);  
s.matches(".\*h.\*e.\*l.\*l.\*o.\*") ? "YES" : "NO"; s.replaceFirst(“[0]\*”,””);  
System.identityHashCode(st); // Memory location of string; same literal values share same location

StringBuilder sb = new StringBuilder(); sb.append(st); sb.length(); sb.replace() sb.reverse(), .delete(st,en+1), .deleteCharAt(), .insert(ind,str), .charAt(), .indexOf(), .setCharAt(), .toString(), .substring(stInd, endInd+1), .capacity()  
StringBuilder[] m = new StringBuilder[n];   
for (int i = 0; i < n; i++) m[i] = new StringBuilder(in.next());

Arrays: int[] a = new int[5]; int a[] = new int[5]; a.length; int[][] b = { {0,1,2}, {3,4,5} };  
new automatically initializes to zero (for numeric), false (for Boolean), and null (for reference)  
int twoD[][] = new int[3][]; twoD[0] = new int[2]; twoD[1] = new int[4]; twoD[2] = new int[8];  
Arrays.sort(a); for (int val : a) {sum += val;} int ind = Arrays.binarySearch(a, findI); //ind<0 nf  
Arrays.fill(a, -1); Arrays.fill(st, “”); Arrays.equals(a, b); i=Arrays.asList(sAr).indexOf(sFind);  
println(Arrays.toString(a)); Variable arguments: main(String… args)

class Node<AnyType> {   
 AnyType data; Node<AnyType> next, previous;   
 public Node(AnyType inputData) this.data = inputData; this.next = null;}  
class LinkedList<E>{ private Node<E> head = null; private Node<E> tail = null; LinkedList(){} }  
By default, a member of a class is public within its own package: data should be private.

Inheritance: class B extends A { super(w,h,d); }  
package MyPackage; import MyPackage; import java.io.\*; import java.util.\*; (java.lang.\* default)  
try { } catch (IOException e) { }

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); br.readLine();  
PrintWriter pw = new PrintWriter(System.out); pw.print(), pw.println()  
PrintWriter pw = new PrintWriter(“filename2”); pw.print(), pw.println();  
Scanner sc = new Scanner(new FileReader(“filename”)); new Scanner(System.in);  
 sc.nextInt(), sc.next(), sc.nextLine(), sc.nextLong(), sc.hasNext(), sc.hasNextInt()

import java.util.Calendar; Calendar cal = Calendar.getInstance(); cal.set(2013, 11, 25); //12/25/13  
System.out.println(cal.get(Calendar.DATE)+" "+cal.get(Calendar.MONTH)+" "+cal.get(Calendar.YEAR));

import java.time.\*; print LocalDate.now(), LocalTime.now(), LocalDateTime.now()  
LocalDate localDate = LocalDate.of(2019, 12, 31); LocalTime localTime = LocalTime.of(10,30);  
LocalDateTime localDateTime = LocalDateTime(localDate, localTime); or (2019, 12, 31, 23, 59, 59, 0)  
ld = ld.plusYears(y).plusMonths(m).minusDays(d); lt= lt.plusHours(h).minusMinutes(m).plusSeconds(s)  
Period period = Period.ofMonths(1); period = period.addDays(d); date = date.plus(period); .munus(p)  
No method chaining of Period, only last counts: Period p = Period.ofDays(1).ofYears(2), only 2 year  
if (dt1.isBefore(dt2)){}; if (dt1.isAfter(dt2)){}; dt1.isEqual(dt2);  
print(dt.getYear +” “+ dt.getMonth() +” “+ dt.getDayOfMonth())  
  
print dt.format(DateTimeFormatter.ISO\_LOCAL\_DATE) ISO\_LOCAL\_DATE\_TIME, ISO\_LOCAL\_TIME  
DateTimeFormatter shortFormatter = DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT);  
.ofLocalizedTime(FormatStyle.MEDIUM), .ofLocalizedDateTime()  
print(shortFormatter.format(dt)); print (dt.format(shortFormatter));  
DateTimeFormatter myFormatter = DateTimeFormatter.ofPattern(“MMMM dd, yyyy, hh:mm”);   
print (dtt.format(myFormatter)); print (myFormatter.format(dtt));  
  
LocalDate dt = LocalDate.parse(“12 31 2019”, myFormatter); LocalTime t = LocalTime.parse(“11:22”);

Import java.util.Formatter; Formatter fmt = new Formatter();   
fmt.format("Formattting %s is easy %d %.2f", "with Java", 10, 98.6); System.out.println(fmt);

System.out.println( String.format("$%.2f", dollarAndCents) ); //Cents rounded to nearest cent

Legacy Classes/Interfaces: Dictionary, Hashtable, Properties, Stack (use ArrayDeque), Vector

ArrayList<String> al = new ArrayList<>(); List<String> list = new ArrayList<>(); List<Integer>   
list.remove(1) //index 1, list.remove(new Integer(1)) //object 1; al.remove(“C”); al.remove(ind),  
al.size(), al.add(“C”); .contains(E), .indexOf(E), al.add(ind,”A2”);.get(ind), .set(ind, E), .isEmpty(), .clear(); list.equals(list2), int ind = Collections.binarySearch(a, findI); //ind<0 nf  
for (String s : al) s2 += s; Collections.sort(al); Coll..reverse(al); import java.util.ArrayList;  
  
ArrayList to Array: String[] ar = al.toArray(new String[0 or al.size()]; Object[] ar = al.toArray()  
Array to ArrayList: List<String> list = Arrays.asList(st1, st2, st3); or Arrays.asList(stArray);  
When copied, both arrays point to same list and are both fixed sized.   
Original/Copied List/Array can be updated with l.set(val) or ar[ind]=val, but can’t add/remove

for (Iterator<Integer> iter = nums.iterator(); iter.hasNext();)  
{ Integer num = iter.next(); System.out.println(num); iter.remove(); } //Print each num & remove  
long stTime= System.currentTimeMillis(); <code>; long totalTime= System.currentTimeMillis()-stTime;

throw new RuntimeException(); // if no proper return value  
try { } catch(exception\_type e) { } finally { }; e.printStackTrace; print(e.getMessage());

LinkedList<E> ll = new LinkedList<E>(); size(), add(E), add(ind, E), get(ind), set(ind, E), contains(E), indexOf(E), clear(); removeFirst() //dequeue; .addLast() //queue; addFirst()  
ArrayDeque<Integer> stack = new ArrayDeque<>(); .push(), .peek(), .pop(), .size(), .isEmpty()  
  
for (int i = 0; i < 10; i++) sum+= ll.get(i);  
ArrayList<Integer>[] al = new ArrayList[n]; for (int i=0;i<n;i++) al[i] = new ArrayList<Integer>();

import java.util.LinkedList;

HashSet<E> hs = new HashSet<>(); size(), add(E), remove(E), contains(E); containsAll(Col); import java.util.HashSet  
TreeSet<E> ts = new TreeSet<>(); size(), add(E), remove(E), contains(E), ceiling(E), floor(E), subset(E1, E2); Iterator<Integer> itr=ts.iterator(); while (itr.hasNext()) out.println(itr.next());  
Set<Integer>[] hs = new HashSet[n]; for (int i = 0; i < n; i++) hs[i] = new HashSet<Integer>();

Iterator iterator = hs.iterator(); while (iterator.hasNext()) out.println(iterator.next());

PriorityQueue<E> pq = new PriorityQueue<E>(); size(), add(E), remove(E), contains(E), poll(),peek()  
for (int i = 0; i < 10; i++) System.out.print(pq.poll() + " "); //Use comparator() to change order  
pq.add(-x); for (int i = 0; i < 10; i++) System.out.print(-pq.poll() + " “); //Descending order

ArrayDeque<E> adq = new ArrayDeque<E>(); size(), add(E), remove(E), contains(E), poll(),peek()  
push(E), pop() //Stack uses push/pop/peek.

HashMap<K,V> hm = new HashMap<K,V>(); hm.put(K,V); containsKey(K), containsValue(V), v=hm.get(K)  
for (Map.Entry<K,V> ent : hm.entrySet()) System.out.println(ent.getKey() + "-" + ent.getValue());  
for (String key : hm.keySet()) if (hm.get(key) > max) { max = hm.get(key); maxDate = key; }  
HashMap<Long,TreeSet<Long>> hm = new HashMap<Long,TreeSet<Long>>(); if (hm.get(n) != null)  
The default capacity is 16. Default fill ratio is 0.75.

TreeMap: sorted in ascending key order  
TreeMap<K,V> tm = new TreeMap<K,V>(); tm.put(K,V); containsKey(K), containsValue(V), v=tm.get(K)  
for (Map.Entry<K,V> ent : hm.entrySet()) System.out.println(ent.getKey() + "-" + ent.getValue());

LinkedHashMap: maintained in order in which they were inserted  
LinkedHashMap <K,V> lm = new LinkedHashMap <K,V>(); lm.put(K,V); containsKey(K), containsValue(V), v=lm.get(K)  
for (Map.Entry<K,V> ent : lm.entrySet()) System.out.println(ent.getKey() + "-" + ent.getValue());  
The default capacity is 16. Default fill ratio is 0.75.

assert isSorted(a); //Statement to test assumptions about your program. Throws exception unless Boolean condition is true;  
 // 1) Helps detect logic bugs; 2) Documents code; Can enable or disable at runtime; java –ea MyProgram; java –da  
 //Can enable or disable at runtime; java –ea MyProgram; java –da MyProgram; Best practices: Use assertions to   
 //check internal invariants; assume assertions will be disabled in production code. Thus, enable when developing

Stack – area of memory where local variables are stored.   
Heap – area of memory referred to by items placed on the stack (e.g. objects).

enum Temperatures : long { Cold = 0, Freezing = 32, Warm = 80, Boiling = 212 } (default is int)  
long x = (long)Tempartures.Cold; (must cast enum)  
enum Party {Democrat, Republican, Liberal} Party myChoice = Party.Republican;  
if (myChoice == Party.Liberal)