Comp 202 Crib Sheet

by Julian Lore Side 1 of 2

Binary first number determines sign, 0001 positive 1, 1111 negative 1 (count backwards)

1 → on	0 → off
001 → 1	010 → 2
011 → 3	100 → 4 . . .
101 → 5	

Binary is base 2

Add 1 to last digit, move to next if passes 1

Bases 5123 in Base 10 = $5(10^3)+1(10^2)+2(10^1)+3(10^0)$ | 11001 in Base 2 = $1(2^4)+2(2^3)+\ldots+1(2^0)$

1.Structure

Braces {Mark the beginning of a block} | Missing/extra {} in beginning or end = compiler error | All { must be in a class/interface

class Like a chapter, holds a lot | public class HelloWorld{}

method Inside classes | public class HelloWorld({public static void *main* (name) String[] args){}}}}|method inside class → method belongs to that class

Main method: Execution of program starts at beginning of main method, need this

Cannot define method inside another

Statements Statements/commands, all end in ;

What you put in () for commands is evaluated

2. Variables

Place in memory reserved for storing a value. Java: give variables **name** and **type**

Why? Store partial results, generalize code, easier to understand

Names can contain letters. numbers, and _ | Names should explain purpose | Convention: first word lowercase, first letter of every word after, uppercase | i.e. degreesInFahrenheit

Can also define your own types of data

Creating var Need 1) Var's type & 2) Var's name → called **declaring** a variable. i.e. int fahrenheitNumber;

Declaring variables important, comp needs to know how much mem to allocate & makes easier for you

Setting value of variable: variable = expression; | i.e. fahrenheit=212; | = in java is an assignment, not an equality sign like in math | expression will be evaluated before assigned

Set value of undeclared var = compiler error

"String literal". takes as string, if you put var in quotes won't take var

!! Variables made in one block **not related** to vars in another block!

Command line arguments public static void main (String[] args)

args is a variable, set by comp when program starts

Type of args is String[] (String array)

1^{st} String accessed by args[0], 2^{nd} String by args[1] . . .

run Test 100 (args[0]) 50 (args[1])

If you parse args that aren't the type parsed → runtime error

3. Types

All data you manipulate has a type, esp. vars

Primitive types These don't reference addresses!

int: Store integers (from -2,147,483,647 up to 2,147,483,647) (32-bit/4 bytes)

(int overflow) 2147483647+1 gives -2147483648, where -2 gives . . . 47

byte: Integers from -128 to 127 (8-bit/1 byte)

short: Int -32,768 to 32767 (16-bit)

long: Int from -2^{63} to $2^{63}-1$ (64-bit)

denoted by L after value (if literal), i.e. 3L is a long

long l=3 does not compile, long l=3L does

double: Store fractions/decimals, not infinite precision, limited decimals

Write .0 after int, get double | i.e. int x = 3 or double x = 3.0 or double x = 3 (gets evaluated to 3.0)

To store a double, computer stores 2 ints, int $1 * 2^{\text{int } 2}$

Rounding issues, limited space

Numbers finite in base10 may be infinite in base-2 i.e. $\frac{1}{10}$

float: Smaller double (32 bit instead of 64 bit)

denoted by f, i.e. 3f is float

float f = 3.0 does not compile, float f=3.0f does

boolean: Store true/false

boolean aceExam = true;

Logical operators for booleans: &&→ and, ||→ inclusive or, !→ not

char: Store single symbol (represented by number)

Literal chars denoted by ', i.e 'M' To store ', write '\''

Reference vs primitive types

Primitive: int x=10; → 1) create space in mem to store int, 2) say that if we use x, we want whatever is there, 3) store 10 in that mem location

Reference: int[]x = {1,2} → 1) create space in mem to store address of array, 2)specify when we use x, want to access space in mem (address), 3)Makes array elsewhere in mem that will store 1,2 and have length 2, must be elsewhere, has address a, 4) set val of x to a (address)

int[] x = {1,2}; int[]y=x; y[0]=2; System.out.printlx(x[0]); Will print 2 because x & y point to same thing, changing something inside what y points to will change it for x too

Swapping

Swap primitive in another method, does not change in method that called it

Swap reference in another, same thing (cannot change address it points to in another method)

But if you change values pointed at in another method, will change values pointed at in method

. . . swap(int[] a, int[] b){int temp = a[0]; a[0]=b[0]; b[0]=temp} Swaps first element of both original arrays but a=b does not change anything in method

String: store multiple symbols

Literal strings stored in , is empty string

Can combine strings with +

Print new line with "Blablabla \ n line lower blabla"

Can use String.length(); to get length (different from arrays, has ())

.charAt(int i);→gets i^{th} symbol of String (count from 0)

.substring(a,b)→gets String from a up to (not incld) b, also .substring(a)→ a to end

.compareTo(anotherString); → Tells which is larger

.indexOf(char c);→ Gives index of first c

.toLowerCase();→ makes new string that is lowercase version

Can define your own type!

If you store type X in something of type Y, compiler error

4.Expressions Can come in several ways:

1) Literal 10, "hello", true, 3.0

2) Variables someVariable

3) Returned val of non-void methods Math.sqrt(8);

4) Combine expressions

Evaluating expressions

Literals evaluate to type of literal

Type of var is what it was **when you declared it**

Operations

Evaluating multiples expression + with each other, go from less inclusive to more inclusive, widening conversion rather than narrowing conv, so no data lost

int+int→int, double+double→double, int+double→double,

String+String→String,

String+int→String

If you want to do a narrowing conv, need to cast

int x = (int) 7.5; → x is 7 (truncates)

Casting temporary, only changes for expression it is in

double y= 3.5, int x = (int) y, y is still a double

double x = (double)1/2 → 1.0/2 → .5

double x = (double)(1/2)→(double)0→0

Order of Operations Like math

parenthesis→Mult,div,modulo→Addition, subtraction→Assignments

Goes from left to right!

"Your number is"+1+2→ "Your number is12"

"Your number is"+(1+2)→"Your number is3"

1+2+"is your number"→ "3is your number"

int x = 10; . . . x=x+1; → x=11

1/2+1/2 → 0 because of trunc

double x = 1/2 → 0, because int divide before assignment

rather: double x = 1.0/2.0 or 1.0/2 or 1/2.0 or 0.5

Other operators -,*,/, % (mod)

Some operators not defined on certain outputs, String*String → Error

INT DIVISION TRUNCATES

9/2 → 4 (decimals cut off! no rounding!)

1/0→ error

Constants: identifier similar to variable, holds on value for entire existence

final double PI = 3.14;

If you assign a value to PI again or declare the same final again, compiler error

5. Creating new methods

2 types of methods, methods someone else wrote (library methods) or methods you wrote

Writing your own allows you to group many commands into 1

Inside a class

Method header is where you give names and stuff for a method like so:

public static **int** **anotherMethod** (**double a**)

int represents output/return type, void if none, var type otherwise. If not void, return blabla; will return to method that called method with value of blabla

return statement has to be reached eventually, if you only have 1 return in an if, then compiler error, because a return has to be able to be reached!

2 ifs, 1 return in each, will not compile! 1 if, 1 else, 1 return in each, will compile, because return will be reached no matter what

Only 1 return statement will be reached during an exec of a method, because method is left once it hits 1^{st} return

method name

Input type and name it will represent inside this method

New method defines new command we can use in program

Some just do things: robot.move();

Whereas others give you values: double x = Math.sqrt(40);

When calling your own methods, don't need to write class (if method belongs to same class as method calling it), as opposed to lib methods

Location of method (before or after current method), does not matter, will scan through whole prog

Remember declaring a var in a method and trying to use it in a diff method → compiler error!

Advantages of methods: code reusability, reduce code dupe, easier debug, problems decomposed, hides tricky logic, easier to read and understand

Disadvantage: a little overhead to set up in beginning (not really)

Modifying given values in another method will not affect the values of the method calling it (unless you assign you return modified value and assign it)

6. if

A block of code that only executes if condition is true

if(condition){ Block, conditional code}

boolean

Want something to happen if true, something else to happen if false. Can use 2 if statements (not efficient/clear/good and both or neither can happen). Instead use

if/else

if(condition){code happens if true}

else{ code happens if false}

Multiple options: if/else if/else

Instead of nesting ifs inside of elses, use

else if

if(condition 1){if true} else if(condition 2){if 1 is false and 2 true} else if(condition 3){if 1,2 false, 3 true} . . . else{all false}

Comp 202 Crib Sheet

by Julian Lore Side 2 of 2

Don't need to end with else for else if
Big difference if we change order of else if
If condition 2 can only be true if condition 1 is true, cond 2 will never be reached
If braces are omitted for if statements, then compiler assumes braces around first statement after condition
Indentation ignored, only for readability ; after if condition → if does nothing i.e. if(x<0); {happens no matter what}
7. boolean
Evaluate to either true or false
boolean expression is either: 1)true or false 2) variable with type boolean 3) call to method that return boolean 4) operator which returns boolean (==,!=,<,>,<=,>=. &&, etc.)

operators

== and != work with any type, inequalities only work with numeric types (or char)
if operans have diff types, lower precision gets promoted, i.e double==int → double==double
&&, ||, ! are boolean operators, take booleans, return booleans
Can combine these together
if(3<x<10) will **not** compile, comp can only do 1 thing at once (3<x)<10→ (true/false)<10 → ???error
Fix: if(3<x && x<10)

Comparing chars

Chars have numerical values, unicode number compared
Note: 26 lowercase letters all in a row, 26 uppercase letters all in a row, so 'a'<'c'

"short-circuit "evaluation

If left of && is false, the remaining operands not looked at
If left of || true, remaining not looked at
Good for speed
if(boolean==true) is useless, if statements already checks if true, so: if(boolean)

Remember x=5 is **not** a boolean! That's an assignment, need ==

Comparing floating pts

Shouldn't use equality operator for float/double
Might think they're close enough, but not exactly equal
.1+.1+.1==.3 → false
Alternatively: get abs val of difference between 2 doubles, say they must be less than some small decimal

Comparing reference types

To compare two ref types, do not use == or !=, as that will compare address! Use a.equals(b);
If you use a==b, you'll get false, if addresses are diff (if both point to 1,2,3, but in diff parts of memory)
i.e. int[] x={1,2}; int[]y=x; x==y→ true but int[] x={1,2}; int[] y={1,2}; x==y→ false

8. Loops

while Execute forever while condition is true
while(condition){Keeps happening until false} Code after will only happen once loop is done, condition is false
Loop counter: int i=0; while(i<5){Do something; i++;} Will do this 5 times
Condition only evaluated at beginning of loop, not after every statement, i.e. can add 10 to i and then subtract 10 from i before loop ends, will still go on
Infinite loops=rip
while(condition); is an infinite loop! Not executing anything

for Common loop theme has 3 steps, initialization, iteration (of loop) given condition, finalization
int x = 0; (initialize) while(x<4)(condition){Loop action; x++; (finalization at end)}
for(initialization;condition; finalization){loop body}
initialization happens once, before first step of loop | condition same as while loop | finalization last step of loop (must be a valid statement! i=i+2 instead of i+2)
Copy of while loop: for(int x=0; x<4; x++){loop action;}
Benefits: more readable

nested loops Loop inside of another loop

9. Arrays
Array is a container object, holds fixed # of vals of 1 type, length fixed & established at creation
Many values of same type into 1 array (1 "object ")
Creating array: type[] arrayName = new type[int of size]
Reserves room in memory, n places for size n, all these places are in a row, all assigned an index
or explicitly declare array entries: type[] arrayName = {1,2,3}

String[] args in main method is an array!
Access entries: arrayName[0], arrayName[1],...
Set values of entries: arrayName[0]=1;
arrayName.length → int giving amount of elements of array

Arrays of Arrays

arrayName[array number][number within array] i.e. a[1][5] gives 6^{th} element of 2^{nd} Array
multidimensionalArray.length → number of arrays contained in array
If you want number of elements in an array, multidimArray[index of array you want].length
Declare multidim array: int[][] arr = {{1,2,3},{1,4}};
or: type[][] name = new type[size 1][size 2]; (makes rectangular array, all same size)
Can also make jagged arrays: type[][] name = new type[5][]; Can put diff size arrays in this
Can have more arrays in arrays type[][][][][][]...

Printing an Array

System.out.println(array); prints out the address in memory

Packages

Can import libraries from packages! Need import java.util.Arrays; in preamble, before class
Packages contain 1+ classes, class contain 1+ methods, methods have 1+ commands
Look up class, gives you package name.
If package ≠ java.lang, need to import
Arrays.toString(int[] x) → {1,2} or whatever its contents are
Null: literal value that can be used for all ref types → points to nowhere
int[] a; by itself does not work, undeclared array. But int[] a=null; does
if a null, a[0] or a.length→RuntimeError (NullPointerException, occurs whenever you use . or [] on null)

10. Exceptions
Impossible to execute → exception or runtime error
Info from exceptions:
Exception in thread "main" **java.util.InputMismatchException** at java.util.Scanner.throwFor (**Scanner.java:819**)
Type of exception
Stack trace, which methods called method that crashed prog
Line number and file that exception occured in

Array exceptions

ArrayIndexOutOfBounds → try to access invalid index, exception gives index number that caused it
NullPointerException (access properties of null var)

Throwing Some commands can't be executed given certain input, instead of using ifs and whatnot to make output null, should throw, problem will become hard to find if not
Immediately generate an error by using throw, rather than hide
if(stuff==null || ...){throw new IllegalArgumentException("Invalid input ")}
Other kinds of exceptions: DivideByZero, NumberFormatException (String to number, etc.)
With throw, method will give value or error
If you want to process the job, use try/catch
try some stuff; catch(IllegalArgumentException e){Happens if that type of exception happens, if no error, skip this, if different type of exception, pass to caller}
Diff types of Exceptions, hierarchy like Arithmetic, etc.
Can use catch(Exception e){ To catch all exceptions} | **Dangerous though!** Can hide all bugs
Can put multiple catch blocks after, first one that matches will be executed
Each method in method chain can catch exception: main()→a()→b()→c()
If c gens exception, can try/catch in b. If not, can try/catch in a, if not, then main. If none catch, then prog crashes
Can also use try/catch/finally, finally will occur no matter what, good for: removing dupe code, clean up before crash/return
Can even do try/finally

3 types of errors Compile time, run time, bug(incld infinite loop) → compile gives most info, run a bit, bug none
Throw used to give runtime error rather than bug
Exception is an object, catch(Exception e) declares var e of type Exception
Can create one via new Exception("info "), make own exception type for particular kind of prob in your code

Commands

Printing

System.out.println(); Prints on new line
System.out.print(); Prints

Parsing

Integer.parseInt(String a); String→ int
Double.parseDouble(String a); String→ double

Math library

Math.sqrt(double a); Returns square root of number
Math.PI(); Approximation of Pi (double)
Math.abs(double a); Returns absolute value of a number
Math.pow(double a, double b); Returns a^b
Math.sin(double a); Returns sin of a (rads)
Math.random(); Returns random double $0.0 \leq double < 1.0$

Misc

++x; or x++→(x=x+1);
–x; or x–→(x=x-1);
x(op,i.e. +)=9 → (x=x(op)9)
Integer.MAX_VALUE; gives integer max value
Integer.MIN_VALUE; gives integer min value

Examples

What's **wrong** with this?

```
int x = Integer.parseInt(args[0]);
if(x<0){
    System.out.println("Positive #")
}
if(x>0){
    System.out.println("Negative #")
}
else{
    System.out.println("0 ")
}
```

If the first if is true and the second false, the else will still print! Else only affects the if above. To fix, change second if to else if

Can't initialize array 2x (even if 1 declares size, the other declares entries)

Misc Var with same name can be declared 2x in same method (i.e. 2 for loops)
Index out of bounds → run time error , printing in a non-void method will still print
You can make two arrays of diff size equal to another, just changes address it points at, int[] a= new int[5]; int[] b = new int[10]; a=b; will work!

Remember negation distributes! !(a || b)→!a&&!b
Bytecode is result of compiling
Constant doesn't need memory
a && b and b&& a don't eval to same result, if first is false, says error, maybe first gives compiler error instead
Compiler error → everything looks fine, runtime error → logic doesn't work
Can compile java file without main method
Default val of int[] is 0 in each pos