Java Shortcuts:

Binary Operator shortcuts:

i += 3 <- shorthand for i = i + 3

This shortcut works for all binary operators:

x /= 10 <- shorthand for x = x / 10

Conditional Operator:

condition ? value1 : value2

- if condition is true, the result is value1. Otherwise the result is value2

Instead of

if (x <= 10)

y = 5;

else

y = -5;

We can write

y = (x <= 10) ? 5 : -1;

Instead of

if (x < 0)

return -1 \* x;

else

return x;

We can write

return x < 0 ? -1 \* x : x;

Switch Statements:

Multiple if statements that test equality of values:

if (month == 1)

days = 31;

else if (month == 2)

days = 28;

else if (month == 3)

days = 31;

switch (month) {

case 1:

days = 31;

break;

case 2:

days = 28;

break;

-or-

switch (month) {

case 1:

case 3:

case 5:

case 7:

case 8:

case 10:

case 12:

days = 31;

break;

case 2:

days = 28;

break;

case 4:

case 6:

case 9:

case 11:

default:

days = 30;

break;

}

Note that break is used to exit execution. If a break is omitted on a case, execution "falls through" to the next case.

Important: the switch statement uses == to test equality. This works well with primitive, and it can work with objects -as long as you are testing whether two objects are the same address-.

(Exception, starting in Java 7, the case will test the contents of String).

For example, the Color class has a bunch of fields pre-set to default colors. Each of these fields stores an Object, and as long as we use those pre-defined colors, we will be comparing objects, not color values:

switch (color) {

case Color.BLACK:

.....

This is also why switch works with the enum type.

Each value of an enum is a specific object, and there is only one object with that value.

switch (month) {

case Month.JAN:

case Month.MAR:

case Month.MAY:

.....

Finally, the Java compiler optimizes string literals. Each occurrence of "hello" in the code is not a separate object. Instead, Java optimizes the code to use the same object.

So, if you are careful in how you design your code, you can use switch on Strings. You need to make sure that the Strings you are comparing are all literals in your code.

For example, "Hi" == "Hi" returns true (though it will be false in the interactions pane as the interactons pane does no optimizations.

On the other hand, if builder is a StringBuilder that stores the characters 'H' and 'i', "Hi" == buffer.toString() will return false.

Increment and Decrement Shortcuts

i++ <- increments i and then returns the original value of i

++i <- increments i by 1 and then returns the new value of i

i-- <- same but with a decrement by 1

--i

Be careful that you use the ++ shortcut correctly! i++ and ++i have different effects.

Ex: i = 5;

i = ++i;

1. i is incremented to 6

2. the value of i is returned and stored into i

The result is that i stores 6.

i = 5;

i = i++;

1. The value of i is returned and remembered for when the right side is done evaluating

2. The value of i is incremented to 6

3. The original remembered value of i is stored into i.

The result is that i stored 5.

Array creation and initialization shortcut.

Ex: int[] a = {1, 2, 3, 4, 5};

Creates an array of 5 ints, and stores the listed values in the array in the order given.

The main method.

The main method is a special method of Java. Every stand-alone program must have a main method. It is what the Java Virtual Machine calls to start your program.

Each class may have one (and only one) main method. The form is

public static void main(String[] args) {

If you have a program that contains a class with a main method, you can run your program from the command line of your computer using the Java runtime environment (JRE):

java DesiredClass

the above can be followed by 0 or more input values that will be passed to your main method through the String array parameter.

(Depending on your system, you may also have to set the CLASSPATH environment property to indicate the folder where the Java files are.)

DesiredClass must have a main method in it.

See the MyFirstProgram class we wrote in the lecture. For example, try

java MyFirstProgram What are you looking at?

We can also create a stand alone program by placing all the classes of the program in a jar file. (A jar file is just a collection of Java .class files, and when creating the

jar file, we specify which class contains the main method that should be called when launching the program.)

You can then run the jar file with

java -jar jarfile

Some operating systems will let you run a jar file by just clicking on it. For others, you can create a script that will launch the program. There are ways of converting java

files into .exe files for Windows, and if you are interested you can search for tools on the web.

Finally, what should you place into the main method to run your program?

The main method is a static method.

The interactions pane is a static context (i.e. it acts like the body of a static method).

So, whatever you would have typed in the interactions pane to launch the program, place into the main method.

Multidimensional Arrays

Java does not have multi-dimensional arrays (neither does C).

Instead, Java allows arrays to store other arrays.

For example, to declare and create an array storing double:

double[] array = new double[10];

To declare and create an array of arrays that stores double:

double[][] table = new double[5][8];

We read the type right to left. The right most [] indicates that we have an array, and what comes before the [] (i.e. "double[]") indicates the type stored in the array.

We read the array creation left to right. What we have is an array of 5 elements, and each element is an array of 8 doubles.

To access an element:

table[3][5] <- again we read left to right. In the first array, we go to the 4th element. That element is an array of double. (We can think of it as the 4th row of the table.)

Then, we go to the 6th element of that array. (We can think of it as going to the 6th column of the array.)

- but if thinking of the array as rows and columns, always remember that there is no "column" stored in memory. We just have an array storing arrays.

Example: loop through the elements of a two dimensional array

int[][] array = new array[10][3];

for (int i = 0; i < array.length; i++) { <- notice that nothing has changed from how we traverse through a "normal" array

for (int j = 0; j < array[i].length; j++) { <- nothing changed here, except that the array we are going to traverse through has its address stored in table[i]

Note that the array shortcut still works. You just need to add a new set of { } for each layer of the array:

int[][] values = {{1, 2, 3}, {4, 5}, {6, 7, 8, 9}};

Creates an array of 3 arrays of int, the first array has length 3, the second has length 2, and the third has length 4.