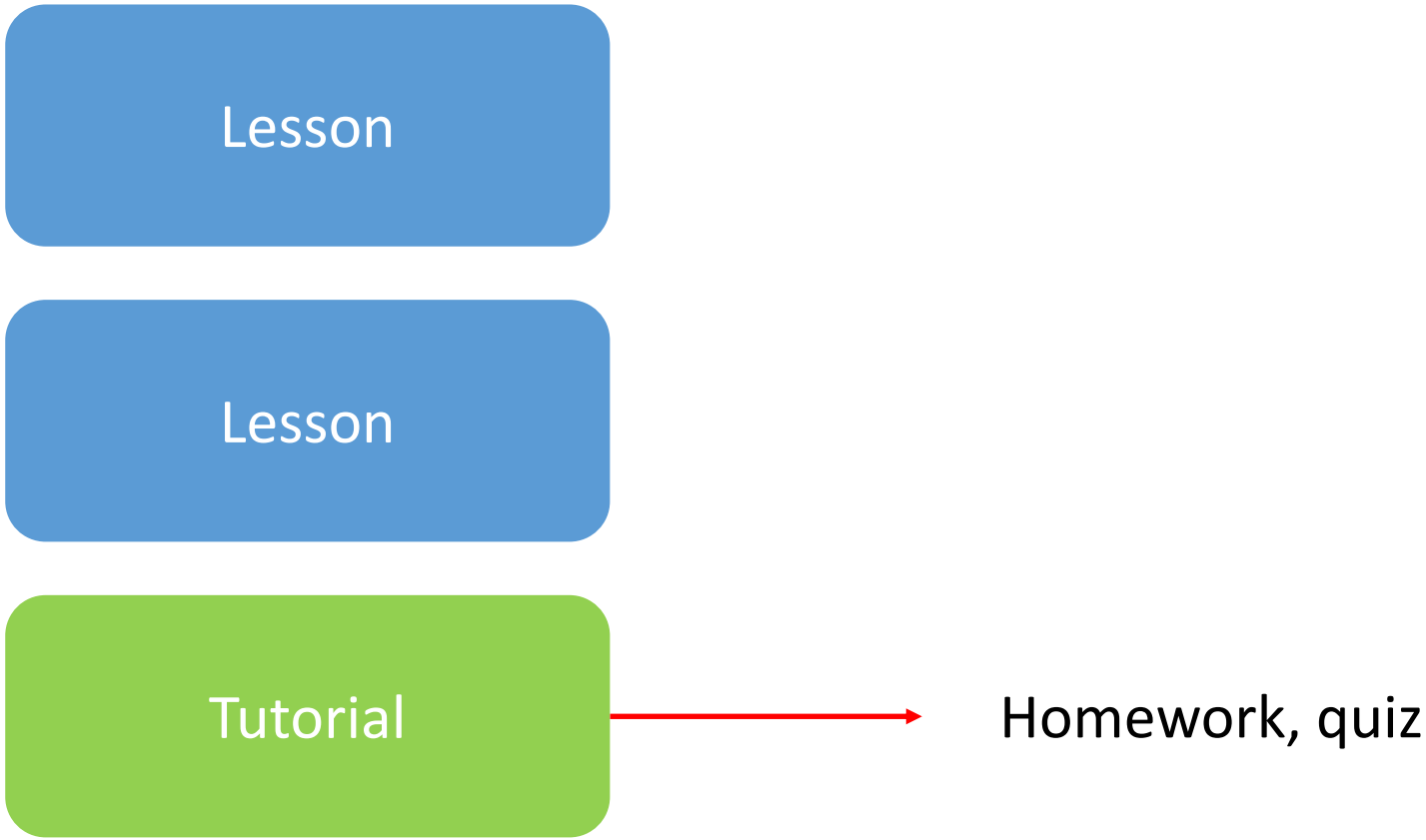
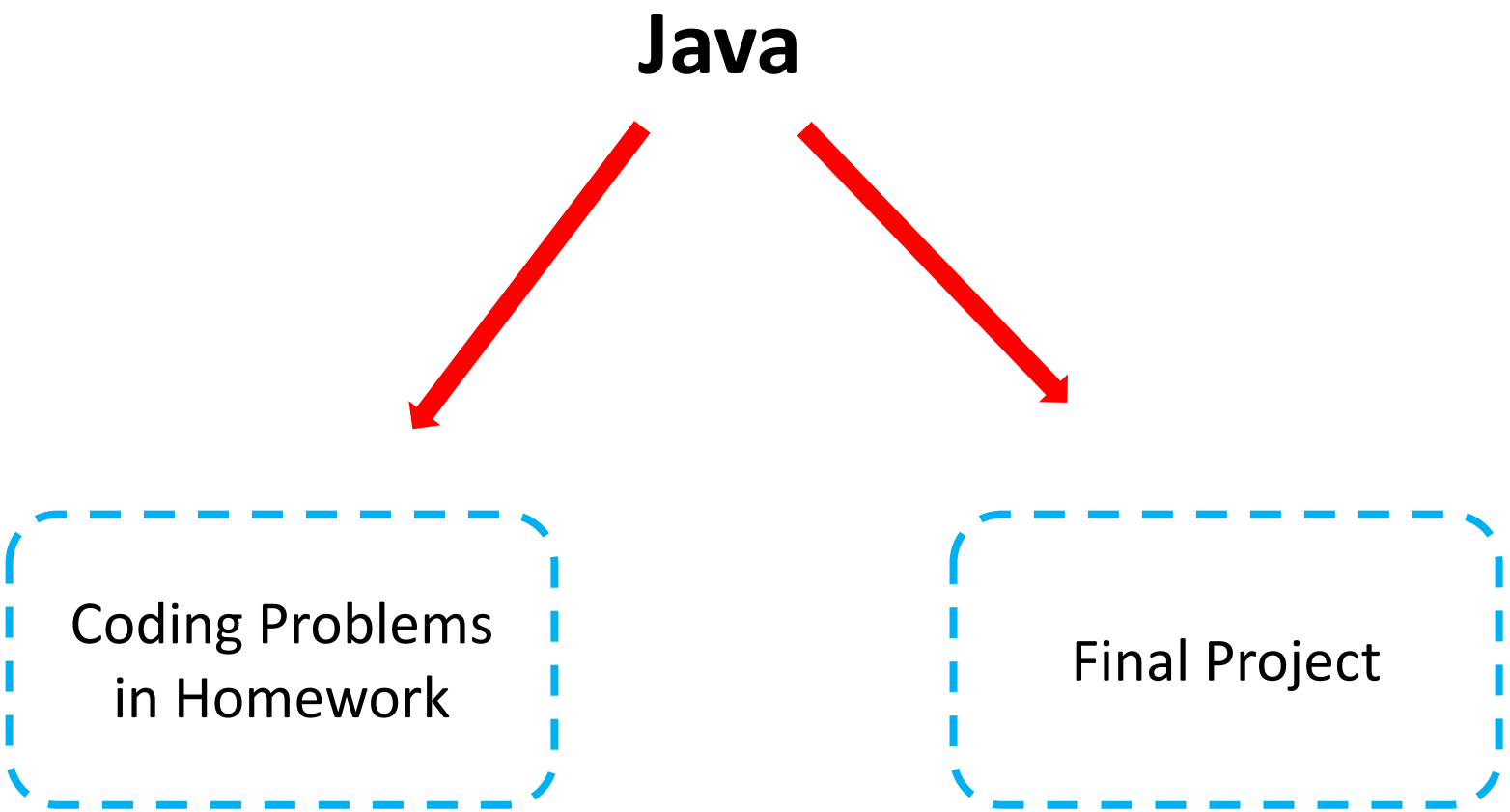
Tutorial

Bran & Yvonne


Lesson
Lesson
Tutorial
Homework, quiz


Java
Coding Problems in Homework
Final Project

Install

https://www.oracle.com/java/technologies/downloads/ (compiler, interpreter and other tools)

IDE

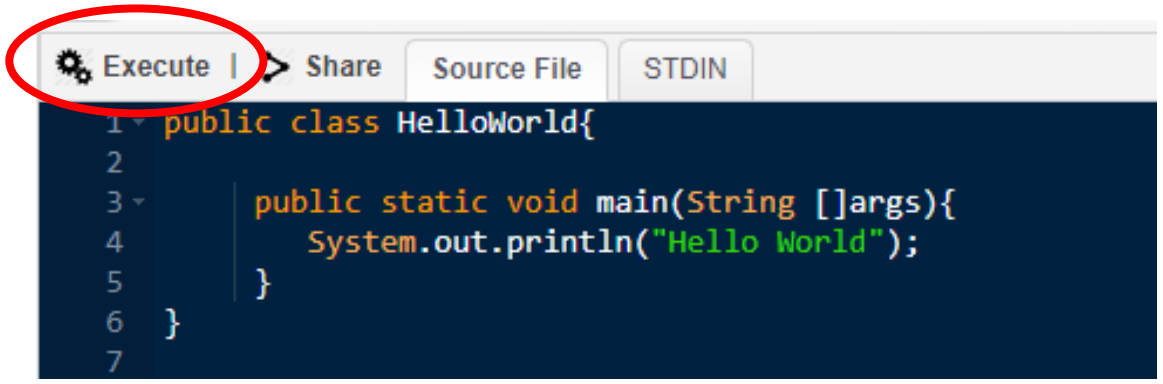
Vscode (simple, lightweight)

IntelliJ (powerful)

Online coding

https://www.tutorialspoint.com/compile_java_online.php

Hello World



HelloWorld.java

Result

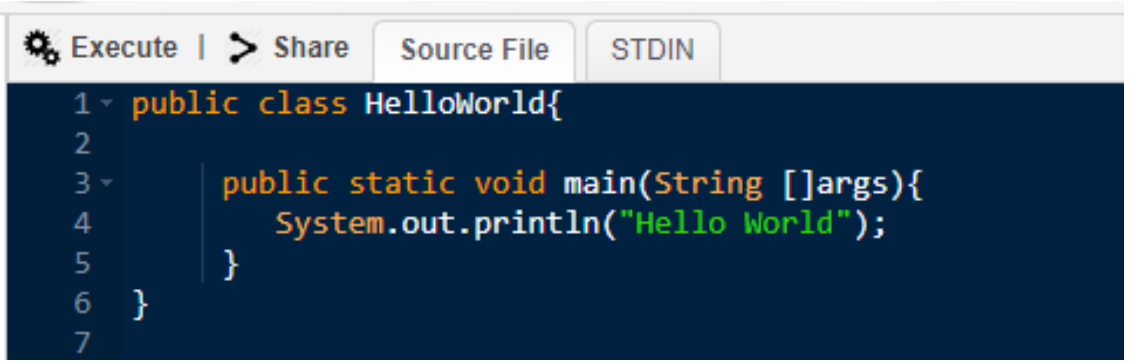
```
$javac HelloWorld.java
$java -Xmx128M -Xms16M HelloWorld
Hello World
```

"**javac**" is the compiler, get HelloWorld.class

"**java**" is the interpreter, run HelloWorld.class

Memory limit, not necessary

Java vs. C++



In Java, every variable, constant, and function (including *main*) must be inside some class.

1.There is no final semi-colon at the end of the class definition.

2.Function *main* is a member of class "HelloWorld" (main is the function where the whole program starts, which is similar to C++)

3.main must: (1) be inside some class. (2) Be public static void.

(3) Have one argument: an array of String. This array contains the command-line arguments. You can use *args.length* to determine the number of arguments (the number of Strings in the array).

Execute | Share | Source File | STDIN

```java
public class HelloWorld{

    public static void main(String []args){
       System.out.println("Hello World");
    }
}
```

System.out.println(...)

System.out.print(...)

Try!

```java
public class HelloWorld{

    public static void main(String []args){
       System.out.print("hello ");    // print a String
       System.out.println(16);         // print an integer
       System.out.println(5.5 * .2);  // print a floating-point number
    }
}
```

Result

```
$javac HelloWorld.java
$java -Xmx128M -Xms16M HelloWorld
hello 16
1.1
```

+ operator

The + operator can be useful when printing.

It is overloaded to work on Strings as follows:

If either operand is a String, it

1.converts the other operand to a String (if necessary)

2.creates a new String by concatenating both operands

```
public class HelloWorld{

    public static void main(String []args){
        int x = 20, y = 10;
        System.out.println("x: " + x + "\ny: " + y);
    }
}
```

Result

```
$javac HelloWorld.java

$java -Xmx128M -Xms16M HelloWorld
x: 20
y: 10
```

Test 1

```java
public class HelloWorld{

    public static void main(String []args){
        int x = 20, y = 10;
        System.out.println(x + y);
        System.out.println(x + y + "!");
        System.out.println("printing: " + x + y);
        System.out.println("printing: " + x * y);
    }
}
```

Result

```
$javac HelloWorld.java

$java -Xmx128M -Xms16M HelloWorld
30
30!
printing: 2010
printing: 200
```

C++ Files vs Java Files

C++

Source Files: .h and .cc (or .cpp or .C)

created by: you (the programmer)
contain : C++ source code
two kinds :
.h (header files)
contain class definitions and function specifications
(just headers - no bodies)
must be included by every file that uses the class / calls the functions
.cc contain implementations of class member functions and "free" functions, including the main function

C++

Object Files: .o

created by: the compiler, when called w/ -c flag; for example:

```
g++ -c main.cc
```

compiles main.cc creating main.o

contain : object code (not executable)

source code is compiled, but not linked/loaded

C++

Executable Files

created by: the compiler (no -c flag)
contain : executable code
Code is compiled if necessary, then linked and loaded.
These are the files that you can actually run, just by typing the name of the file.
name : default = a.out
any other name is possible via the -o flag; for example:
g++ main.o -o test
creates an executable named "test"

Java

Source Files: .java

created by : you (the programmer)
contain : Java source code (one or more classes per file)
restrictions :
(1) each source file can contain at most one *public* class
(2) if there is a public class, then the class name and file name must match

Examples

If a source file contains the following:

```
public class Test { ... }
class Foo { ... }
class Bar {... }
```

then it must be in a file named Test.java

If a source file contains the following:

```
class Test { ... }
class Foo { ... }
class Bar {... }
```

then it can be in any ".java" file

Java

Bytecode Files: .class

created by: the Java compiler
contain : Java bytecodes, ready to be "executed" -- really interpreted -- by the Java interpreter
names : for each class in a source file (both public and non-public classes), the compiler creates one ".class" file, where the file name is the same as the class name

Example

If a source file contains the following:

```
public class Test { ... }
class Foo { ... }
class Bar {... }
```

then after compiling you will have three files:

```
Test.class
Foo.class
Bar.class
```

Test 2

Write a complete Java program that uses a loop to sum the numbers from 1 to 10 and prints the result like this:

```
The sum is: xxx
```

Note: Use variable declarations, and a *for* or *while* loop with the same syntax as in C++.

Test 2

```java
public class HelloWorld{

    public static void main(String []args){
        int sum=0;
        for (int i = 1; i <= 10; i++) {
            sum += i;
        }
        System.out.print("The sum is: " + sum);
    }
}

```

Result

```
$javac HelloWorld.java
$java -Xmx128M -Xms16M HelloWorld
The sum is: 55
```

Java Types

In Java, there are two "categories" of types: *primitive types* and *reference* types:

Primitive Types		**Reference Types**
boolean	same as bool in C++	arrays
char	holds one character	classes
byte	8-bit signed integer	
short	16-bit signed integer	
int	32-bit signed integer	
long	64-bit signed integer	
float	floating-point number	
double	double precision floating-point number	

Notes:

1. no struct, union, enum, unsigned, typedef
2. arrays and classes are really pointers!!

C++ Arrays vs Java Arrays

In C++, when you declare an array, storage for the array is allocated. In Java, when you declare an array, you are really only declaring a pointer to an array; storage for the array itself is not allocated until you use "new":

C++

```
int A[10];  // A is an array of length 10
A[0] = 5;   // set the 1st element of array A
```

JAVA

```
int [] A;           // A is a pointer to an array
A = new int [10];   // now A points to an array of length 10
A[0] = 5;           // set the 1st element of the array pointed to by A
```

C++ Arrays vs Java Arrays

In both C++ and Java you can initialize an array using values in curly braces. Here's example Java code:

```
int [] A = {1, 222, 0};  // A points to an array of length 3
                         // containing the values 1, 222, and 0
```

C++ Arrays vs Java Arrays

In Java, a default initial value is assigned to each element of a newly allocated array if no initial value is specified. The default value depends on the type of the array element:

Type	Value
boolean	false
char	'\u0000'
byte, int, short, int, long, float, double	0
any pointer	null

C++ Arrays vs Java Arrays

In Java, an out-of-bounds array index always causes a runtime error.

In Java, you can determine the current length of an array (at runtime) using ".length":

```
int [] A = new int[10];
    ...  A.length ...          // this expression evaluates to 10
    A = new int[20];
    ...  A.length ...          // now it evaluates to 20
```

Test 3

Write a Java function called NonZeros, using the header given below. NonZeros should create and return an array of integers containing all of the non-zero values in its parameter A, in the same order that they occur in A.

```
public static int[] NonZeros( int [] A )
```

Test NonZeros function in main function. Print the passed arrays and returned arrays as follow:

```
passing [0, 1, 2, 3, 2] got back [1, 2, 3, 2]
```

Test 3

```java
public class HelloWorld{
  public static int[] NonZeros( int[] A ) {
    // count # nonzero values
      int nonz = 0;
      for (int k=0; k<A.length; k++) if (A[k] != 0) nonz++;

    // allocate and fill new array
      int[] result = new int[nonz];
      int j = 0; // index of next element of new array to fill in
      for (int k=0; k<A.length; k++) {
         if (A[k] != 0) {
            result[j] = A[k];
            j++;
         }
      }
      return result;
  }

  public static void PrintArray( int[] A ) {
    System.out.print("[");
    for (int k=0; k<A.length; k++) {
        System.out.print(A[k]);
        if (k < (A.length - 1)) System.out.print(" ");
    }
    System.out.print("]");
  }

  public static void main(String[] args) {
    int[] A = {0,1,2,3,2};
    System.out.print("passing ");
    PrintArray(A);
    A = NonZeros(A);
    System.out.print(" got back ");
    PrintArray(A);
    System.out.println();
  }
}
```

Result

```
$javac HelloWorld.java
$java -Xmx128M -Xms16M HelloWorld
passing [0 1 2 3 2] got back [1 2 3 2]
```

Arraycopy

In Java, you can copy an array using the *arraycopy* function. Like the output function *println*, arraycopy is provided in java.lang.System, so you must use the name System.arraycopy. The function has five parameters:

src: the source array (the array from which to copy)
srcPos: the starting position in the source array
dst: the destination array (the array into which to copy)
dstPos: the starting position in the destination array
count: how many values to copy

Arraycopy

Here's an example:

```
int [] A, B;
A = new int[10];
 -- code to put values into A --
B = new int[5];
System.arraycopy(A, 0, B, 0, 5)  // copies first 5 values from A to B
System.arraycopy(A, 9, B, 4, 1)  // copies last value from A into
                                 // last element of B
```

Arraycopy

- The destination array must already exist and it must be large enough to hold all copied values.
- The source array must have enough values to copy (i.e., the length of the source array must be at least srcPos+count).
- For arrays of primitive types, the types of the source and destination arrays must be the same (so for example, you cannot copy from an array of int to an array of double or vice versa).
- For arrays of non-primitive types, System.arraycopy(A, j, B, k, n) is OK if the assignment B[0] = A[0] would be OK.

Arraycopy

The arraycopy function also works when the source and destination arrays are the *same* array; so for example, you can use it to "shift" the values in an array:

```java
public class HelloWorld{
    public static void main(String []args){
        int [] A = {0, 1, 2, 3, 4};
        System.arraycopy(A, 0, A, 1, 4);
        PrintArray(A);
    }

    public static void PrintArray( int[] A ) {
        System.out.print("[");
        for (int k=0; k<A.length; k++) {
            System.out.print(A[k]);
            if (k < (A.length - 1)) System.out.print(" ");
        }
        System.out.print("]");
    }
}
```

Result

```
$javac HelloWorld.java
$java -Xmx128M -Xms16M HelloWorld
[0 0 1 2 3]
```

Multidimensional

As in C++, Java arrays can be *multidimensional*. For example, a 2-dimensional array is an array of arrays. Two-dimensional arrays need not be rectangular. Each row can be a different length. Here's an example:

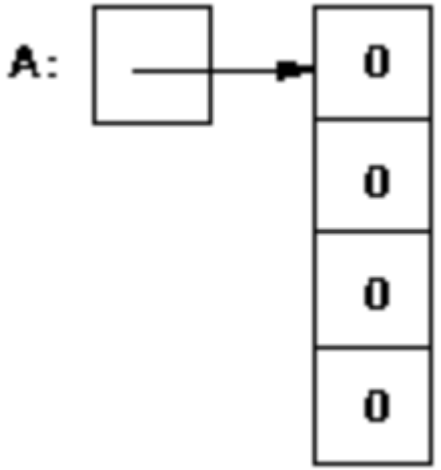
```
int [][] A;           // A is a two-dimensional array
A = new int[5][];     // A now has 5 rows, but no columns yet
A[0] = new int [1];   // A's first row has 1 column
A[1] = new int [2];   // A's second row has 2 columns
A[2] = new int [3];   // A's third row has 3 columns
A[3] = new int [5];   // A's fourth row has 5 columns
A[4] = new int [5];   // A's fifth row also has 5 columns
```

Test 4

Draw the value of “A”. If run error, just write “error”. Here are two examples:

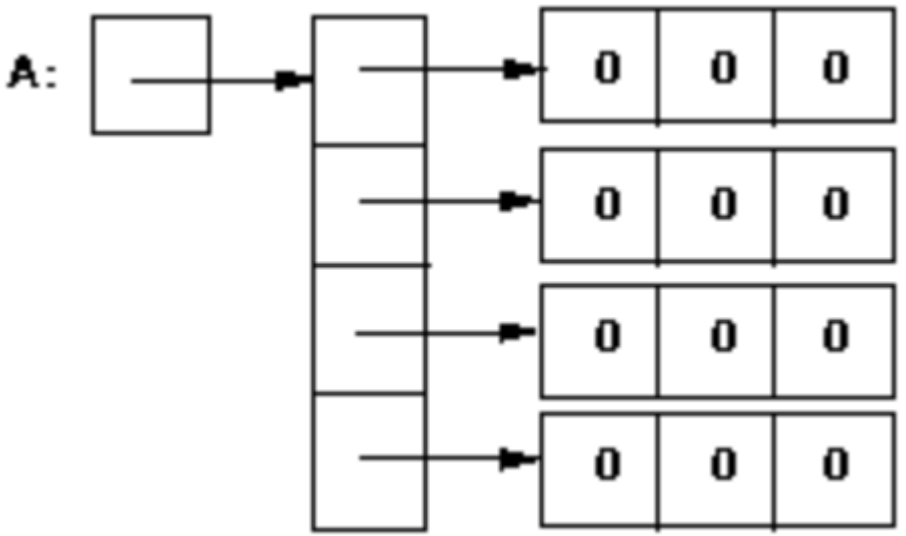
int [] A;

int [] A = new int [4];

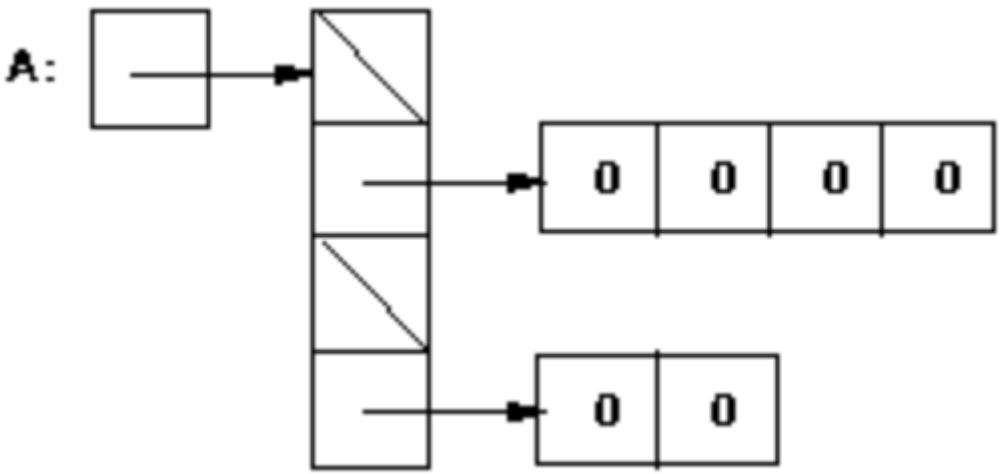


Test 4

```
int [][] A = new int[4][3];
```

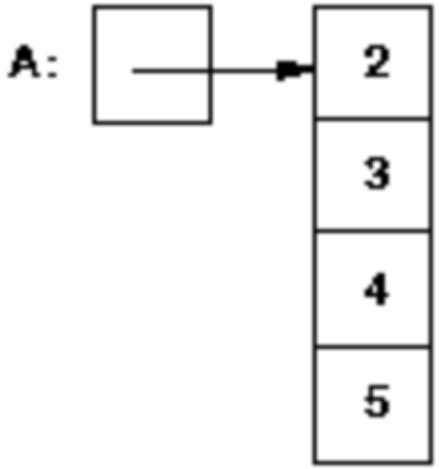



```
int [][] A = new int[4][];
A[1] = new int[4];
A[3] = new int[2];
```



Test 4

```
int [] A = new int[4];
int [] B = {0,1,2,3,4,5,6,7,8,9};
System.arraycopy(B,2,A,0,4);
```




```
int [] A = new int[4];
int [] B = {2,3,4};
System.arraycopy(B,0,A,0,4);
```

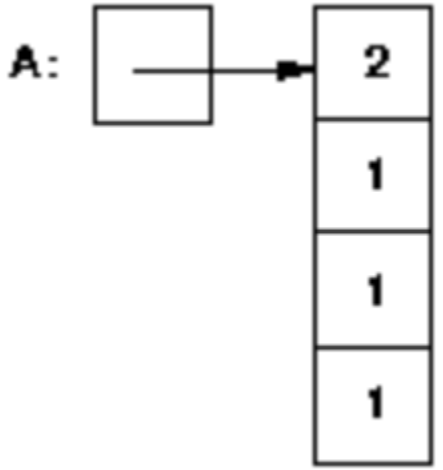
error

Test 4

```
int [] A = new int[4];
int [] B = {0,1,2,3,4,5,6,7,8,9};
System.arraycopy(B,8,A,0,4);
```

error

```
int [] A = {1,1,1,1};
int [] B = {2,2,2};
System.arraycopy(A,0,B,1,2);
System.arraycopy(B,0,A,0,3);
```



C++ Classes vs Java Classes

In C++, when you declare a variable whose type is a class, storage is allocated for an object of that class, and the class's constructor function is called to initialize that instance of the class.

In Java, you are really declaring a pointer to a class object; no storage is allocated for the class object, and no constructor function is called until you use "new".

Assume that you have defined a List class as follows:

```
class List {
  public void AddToEnd(...)
  { ...}
  ...
}
```

C++

```
List L;            // L is a List; the List constructor function is called to
                   // initialize L.
List *p;           // p is a pointer to a List;
                   // no list object exists yet, no constructor function has
                   // been called
p = new List;      // now storage for a List has been allocated
                   // and the constructor function has been called
L.AddToEnd(...)    // call L's AddToEnd function
p->AddToEnd(...)   // call the AddToEnd function of the List pointed to by p
```

Assume that you have defined a List class as follows:

```
class List {
  public void AddToEnd(...)
  { ...}
  ...
}
```

JAVA

```
List L;          // L is a pointer to a List; no List object exists yet
L = new List(); // now storage for a List has been allocated
                // and the constructor function has been called;
                // note that you must use parentheses even when you are not
                // passing any arguments to the constructor function
L.AddToEnd(...) // no -> operator in Java -- just use .
```

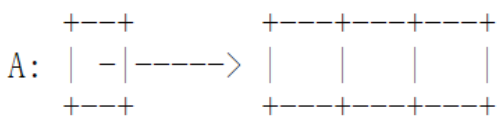
Aliasing Problems in Java

The fact that arrays and classes are really pointers in Java can lead to some problems:

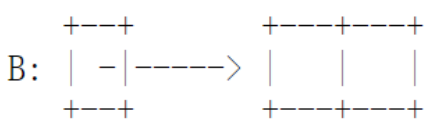
Java code

conceptual picture
(all empty boxes contain zeros)

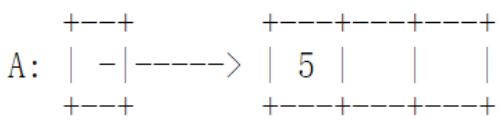
```
int [] A = new int[3],
```



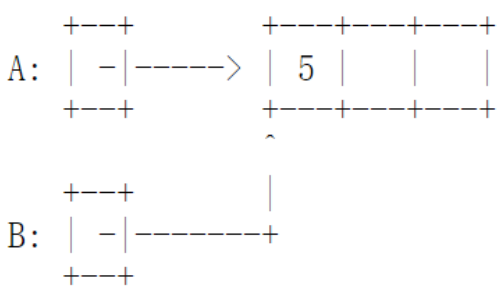

```
       B = new int[2];
```



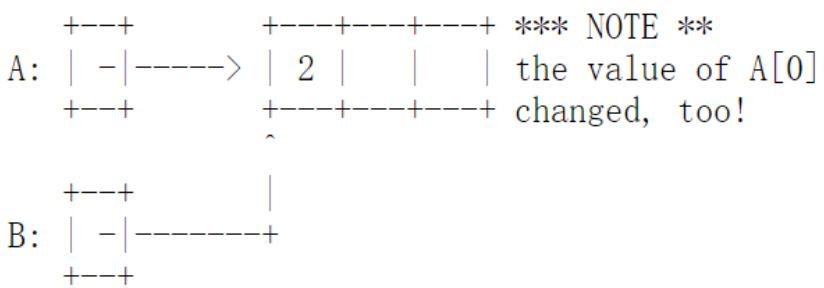

```
       A[0] = 5;
```




```
       B = A;
```




```
        B[0] = 2;
```



```
void f( int [] A )
{
    A[0] = 10;   // change an element of parameter A
    A = null;    // change A itself
}

void g()
{
    int [] B = new int [3];
    B[0] = 5;
    f(B);
    *** B is not null here, because B itself was passed by value
    *** however, B[0] is now 10, because function f changed the first element
    *** of the array
}
```

Aliasing Problems in Java

Solution: *arraycopy* or class's *clone* operation

Test 5

For each of the following Java code fragments, say whether it causes a compile-time error, a run-time error, or no error. If there is an error, explain why.

1.
```
int A[5];
```

2.
```
int [] A, B;
B = 0;
```

3.
```
int [] A = {1, 2, 3};
int [] B;
B = A;
```

4.
```
int [] A;
A[0] = 0;
```

5.
```
int [] A = new int[20];
int [] B = new int[10];
A = B;
A[15] = 0;
```

Test 5

```
int A[5];
```

Compile-time error: Can't specify array dimension in a declaration. This is C/C++ syntax.

```
int [] A, B;
B = 0;
```

Compile-time error: Incompatible type for =. Can't convert int to int[]. B is an array reference, not an int, and 0 is not equiv to null as in C/C++.

```
int [] A = {1,2,3};
int [] B;
B = A;
```

No errors.

Test 5

```
int [] A;
A[0] = 0;
    Compile-time error: Variable A may not have been initialized.
    The array was never allocated.
```

```
int [] A = new int[20];
int [] B = new int[10];
A = B;
A[15] = 0;
    Runtime error: ArrayIndexOutOfBoundsException: 15
    A now references the same array as B, which only has length 10
```

Type Conversion

Type conversion

Java is much more limited than C++ in the type conversions that are allowed. Here we discuss conversions among primitive types. Conversions among class objects will be discussed later.

Booleans cannot be converted to other types.

For the other primitive types (char, byte, short, int, long, float, and double), there are two kinds of conversion: *implicit* and *explicit*.

Implicit conversion

An implicit conversion means that a value of one type is changed to a value of another type without any special directive from the programmer.

A char can be implicitly converted to an int, a long, a float, or a double.

For example, the following will compile without error:

```
char c = 'a';
int k = c;
long x = c;
float y = c;
double d = c;
```

Implicit conversion

For the other (numeric) primitive types, the basic rule is that implicit conversions can be done from one type to another if the range of values of the first type is a subset of the range of values of the second type.

For example,

a byte can be converted to a short, int, long or float;

a short can be converted to an int, long, float, or double, etc.

Explicit conversion

Explicit conversions are done via ***casting***: the name of the type to which you want a value converted is given, in parentheses, in front of the value.

```
double d = 5.6;
int k = (int)d;
short s = (short)(d * 2.0);
```

Casting can be used to convert among any of the primitive types except boolean.

casting can lose information; for example, floating-point values are truncated when they are cast to integers (e.g., the value of k in the code fragment given above is 5)

casting among integer types can produce wildly different values (because upper bits, possibly including the sign bit, are lost)

Test 6

Fill in the table below as follows:

- If the declaration will compile as is, put a check in the second column, and write the value of the declared variable in the last column.

- If the declaration will not compile as is, but can be made to compile by adding an explicit cast, rewrite the declaration with the correct explicit cast in the third column, and write the value of the declared variable in the last column.

- If the declaration will not compile, and cannot be fixed by adding an explicit cast, put a check in the fourth column.

Test 6

Declaration	*Correct*	*Rewrite with cast*	*Never correct*	*Variable's value*
double d = 5;	X			5.0
int k = 5.6;				
long x = 5.4;				
short n = 99999;				
int b = true;				
char c = 97;				
short s = -10.0;				

Test 6

Declaration	Correct	Rewrite with cast	Never correct	Variable's value
double d = 5;	X			5.0
int k = 5.6;		int k = (int) 5.6		5
long x = 5.4;		long x = (long) 5.4		5
short n = 99999;		short n = (short) 99999		-31073
int b = true;			X	
char c = 97;	X			'a'
short s = -10.0;		short s = (short) -10.0		-10

Reference

https://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial/

Final Project

Goal

In this project, you are required to implement an **interpreter** for the programming language ***SimPL***(pronounced simple). ***SimPL*** is a simplified dialect of ML, which can be used for both **functional** and **imperative** programming.

Principles

Syntax, Names, Types, Semantics

```
let add = fn x => fn y => x + y
in  add 1 2
end
(* ==> 3 *)
```

Thansk