CS 251
Intermediate Programming
Java Basics

Brooke Chenoweth

University of New Mexico

Spring 2024

Prerequisites

These are the topics that I assume that you have already seen:

- Variables
- Boolean expressions (Conditions)
- If and switch statements
- Loops (for/while)
- Arrays
- Methods / Functions / Procedures

In *any* language (C, C++, Matlab, Java)

How to program

- Program Design Decisions
- Write / Compile / Execute
- **Test your program**
  - Find errors!!!
- Read manuals
  - The book (or other reference books)
  - Java Tutorials
  - Java API

Programming Style

- Keep it Short and Simple
- Be CONSISTENT
- Comment your code (javadoc)!
- Use MEANINGFUL variable names
- Use proper indentation (editor can help you)
- Java code conventions
- Javadoc conventions

Program Design

- Break down the program in parts that you can handle
  - Identify parts of a program
  - How to solve various parts (algorithms)
- Sketch out what the program should look like
  - Draw an object diagram
  - Write some pseudo code

Java for non-java progammers

If you have not previously seen Java, the start of this class will be a bit steep. Remember, there's plenty of help available - use it!

Here are a few things to remember:

- Java is not an interpreted language
- Java contains thousands of predefined classes
- The syntax may be different, but the concepts listed above still apply

The very basics

- Comments (one-line?, multi-line?)
- Program Statements (valid?, invalid?)
- Blocks (braces?, no braces?)
- Variables (simple?, instantiations?), Variable names
- Loops (for, while, do...while)
- Initializations?
- Assignments (valid?, invalid?)
- Constants

Comments

Why comment your code?

- Explain tricky code passages
- Put notes for yourself
- In large projects, comments are paramount (Why?)
- Java provides a facility called Javadoc – We'll be using it.
- No need to comment every line (Why?)

Java Comments

Java has three different types of comments:

- **One liner** - `// Here's my one line comment`
  Used to make notes and explain local variables
- **Multi liner** - `/* Several lines of comments */`
  When using more than one line, this is often done
- **Javadoc** - `/** Javadoc formatted comment here */`
  Much more on this later

Program statements

A program statement is usually a line of code that may be (or not) followed by a delimiter of some sort, that indicates one step in program execution.

Several statements make a program.

Java Statements

- Most Java statements are followed by a semicolon – ';'
- Example 1: `int x = 5;`
- Example 2: `System.out.println("Hello!");`
- Some statements (like if statements and loops), are not followed with a semicolon

Program blocks

A program block is a number of statements that have somehow been grouped together, this might be in a method, in a loop, or by themselves.

Java blocks are surrounded by curly braces { }, or in case where there's just one line in a block, the braces can be left out. (Be careful with that!)

Variables

- Are used to hold pieces of information
- Refers to a location in memory
- Should have descriptive names
- Can have different *data types*

Java Variables

Java is what's called a *strictly typed* language. It's like math where you have to have all units match up in a calculation. Java types have to match up as well.

- All java variables have a specific type
- Converting between the types can be done with type casting (if allowed)
- Can have arbitrarily long variable names (use with caution)
- Must be declared before usage

Bits, Bytes, and Prefixes

- 1 bit = 1 or 0, on or off, true or false
- 1 byte = 8 bits ($2^8 = 256$ states)
- Prefixes: k = 10^3, M = 10^6, G = 10^9, T = 10^{12}, P = 10^{15}
- Computers: k = 2^{10}, M = 2^{20}, G = 2^{30}, T = 2^{40}, P = 2^{50}

Cheating companies: 80GB harddrive = 76.3GB
Hard drive specs are given in decimal MB...
New set of binary prefix may help with ambiguity:
kibibyte, mebibyte, gibibyte (KiB, MiB, GiB)
(...if they ever catch on)

Java Data Types

There are two different types of data types:

- **Primitive** - These contain only the data they indicate. There are 8 primitive data types: `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`. They vary in the amount of information they can keep.
- **Reference types** - Are all other types of variables, they hold references to objects, similar to pointers in C and C++.

Java Primitive Data Types

- **boolean** - Truth values, i.e., `true` or `false` - 1 bit
- **char** - Unicode Characters (2 bytes)
- **byte** - One byte
- **short** - Short integer (2 bytes)
- **int** - Regular integer (4 bytes)

$$(-2^{31} \Longrightarrow (2^{31} - 1))$$

- **long** - Large integer (8 bytes)
- **float** - Floating point (4 bytes)
- **double** - Floating point (8 bytes)

$$(2^{-1074} \Longrightarrow (2 - 2^{-52}) \cdot 2^{1023})$$

Declaring Java Variables

On the general form it's:

`<data type> <name>;`

Some examples:

```
int x = 10;
double pi = 3.14159265;
char c;
```

Type casting

Means to "promote" or "demote" one type to another, but exercise caution because you can lose precision, hence there are two types of casts:

- Implicit – Safe casts, Java takes care of it automatically, ex. casting a `byte` to an `int`
- Explicit – Unsafe casts, Programmer must specify the cast.

Examples (supposing variables on prev page):

- Implicit: `pi = x; //Ok since int fits in double`
- Explicit: `x = (int)pi; //Explicit, decimals lost`

Java `boolean` datatype

```
boolean isOpen = true;
boolean isClosed = false;
```

`true` and `false` are *reserved keywords* in Java.

Boolean Expressions and Conditions

Often need truth values, i.e.,

- If some condition is satisfied, do this
- Programming representation of a flow chart
- Loop conditions
- Exists in all programming languages

Boolean Expressions

Basic operations such as:

- And – &&
- Or – ||
- Not – !

Truth tables! What do they look like?

A	B	A and B	A or B
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

A	not A
F	T
T	F

Java Flow Control

If and switch statements, and three (or four) types of loops:

- while
- do. . . while
- for
- Enhanced for loops

An if statement

```
public class IfExample {
  public static void main (String[] args ) {
    int myNumber = 10;
    if ( myNumber < 0 || myNumber > 100 ) {
      System.out.println ( "Invalid Number" );
    } else {
      System.out.println ( "You chose " + myNumber );
    }
  }
}
```

The switch statement

```
public class SwitchExample {
  public static void main ( String[] args ) {
    int myNumber = 10;
    switch ( myNumber ) {
      case 1: case 2: case 3:
        System.out.println ( "You got 1, 2, or 3" );
        break;
      case 5:
        System.out.println ( "You got 5" );
        break;
      default:
        System.out.println ( "Some other number" );
    }
  }
}
```

Example while loop

```
public class WhileLoopExample {
  public static void main ( String[] args ) {
    int j = 100, sum = 0;
    while ( j > 0 ) {
      sum += j;
    }
    System.out.println ( "Sum is: " + sum );
  }
}
```

Example while loop

```
public class WhileLoopExample {
  public static void main ( String[] args ) {
    int j = 100, sum = 0;
    while ( j > 0 ) {
      sum += j;
    }
    System.out.println ( "Sum is: " + sum );
  }
}
```

- But... There's something wrong here?

Example while loop...

We probably want to change the variable we are testing.

```
public class WhileLoopExample {
  public static void main ( String[] args ) {
    int j = 100, sum = 0;
    while ( j > 0 ) {
      sum += j--;
    }
    System.out.println ( "Sum is: " + sum );
  }
}
```

Example while loop...

We probably want to change the variable we are testing.

```
public class WhileLoopExample {
  public static void main ( String[] args ) {
    int j = 100, sum = 0;
    while ( j > 0 ) {
      sum += j--;
    }
    System.out.println ( "Sum is: " + sum );
  }
}
```

Brings us to post/pre decrement/increment operators

- `j++, --j`
- How do these work?

Increment and Decrement

- The `++` and `--` operators add/subtract 1 from a variable and reassign the variable to that value.
- `x++` or `++x` are shorter to type than `x += 1` or `x = x + 1`
- What's the difference between `++x` and `x++` ?

Increment and Decrement

- The `++` and `--` operators add/subtract 1 from a variable and reassign the variable to that value.
- `x++` or `++x` are shorter to type than `x += 1` or `x = x + 1`
- What's the difference between `++x` and `x++` ?
  - The difference is noticable when using the value of the expression.
  - Preincrement: `++x`
    Increments `x`, returns the new value
  - Postincrement: `x++`
    Saves value of `x`, increments `x`, returns the old value
  - Use as separate statement and it won't matter.

Example do-while loop

```
public class DoWhileLoopExample {
  public static void main ( String[] args ) {
    Scanner sc = new Scanner ( System.in );
    int num = 0;
    do {
      System.out.print ("Enter a number [1-100]: ");
      num = sc.nextInt();
    } while ( num < 1 || num > 100 );
  }
}
```

Example for loop

```
public class ForLoopExample {
  public static void main ( String[] args ) {
    for ( int i = 0; i < 10; i++ ) {
      System.out.println ( "i = " + i );
    }
  }
}
```

Note, the curly braces on the loop can be left out if there's only one statement in the loop. (This is also true for other loops and if statements.)
Please don't! It will come back to bite you when you add a second statement.

Example enhanced for loop

```
public class ForLoopExample {
  public static void main(String[] args) {
    for(String arg : args) {
      System.out.println(arg);
    }
  }
}
```

Enhanced for loops let you iterate over a collection or array without manually managing an index variable. Very handy!

Why Arrays?

- Do you remember your math?

$$\sigma = \frac{\sum_{i=1}^{N} (x_i - \overline{x})}{N - 1}$$

What is this?

Why Arrays?

- Do you remember your math?

$$\sigma = \frac{\sum_{i=1}^{N} (x_i - \overline{x})}{N - 1}$$

  What is this?
- Right... The standard deviation...
- So, if you have N variables, of the same type, but different values, you need N variable declarations in order to store those values.
- And in a loop, a way of accessing all those variables in order

What is an array?

- An array is basically an indexed variable, just like the formula on previous slide.
- Array indices always start at 0 (zero). Java array are 0-based arrays.
- The number of elements in the array can be accessed through by reading the `length` variable in the object.
- That's right, a Java array is an object

Array declaration

- The standard form is:

  ```
  <type>[] <variableName>;
  ```

- You can declare arrays of *any* type you want

But... The above doesn't tell you how many elements there should be in the array

Array declaration

- The standard form is:

  ```
  <type>[] <variableName> = new <type>[<size>];
  ```

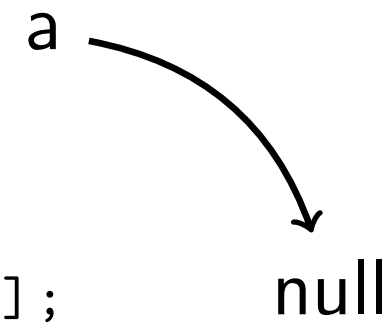
- The `size` tells us how many elements are in that array
- Arrays are initialized by default (on creation), this means:
  - Arrays of numbers contain all 0's
  - Arrays of reference types contains all `null`
- If you didn't create the array, you can still find out the length of it by using the `<variableName>.length` expression
  - This means, access the `length` instance variable in the array object referred to by the variable `<variableName>`.

Array declaration

```
int[] a;

int[] b = new int[4];

String[] c = new String[3];
```

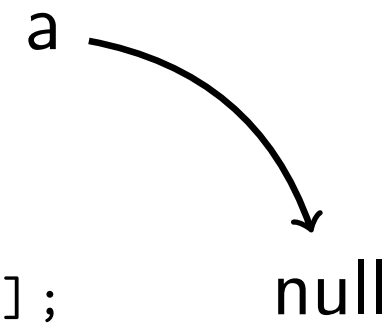
Array declaration

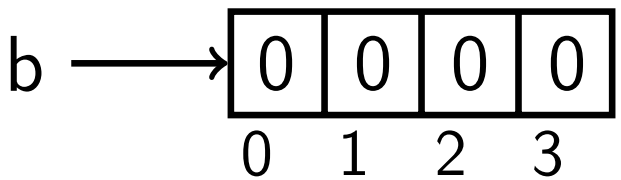
```
int[] a;

int[] b = new int[4];

String[] c = new String[3];
```



Array declaration

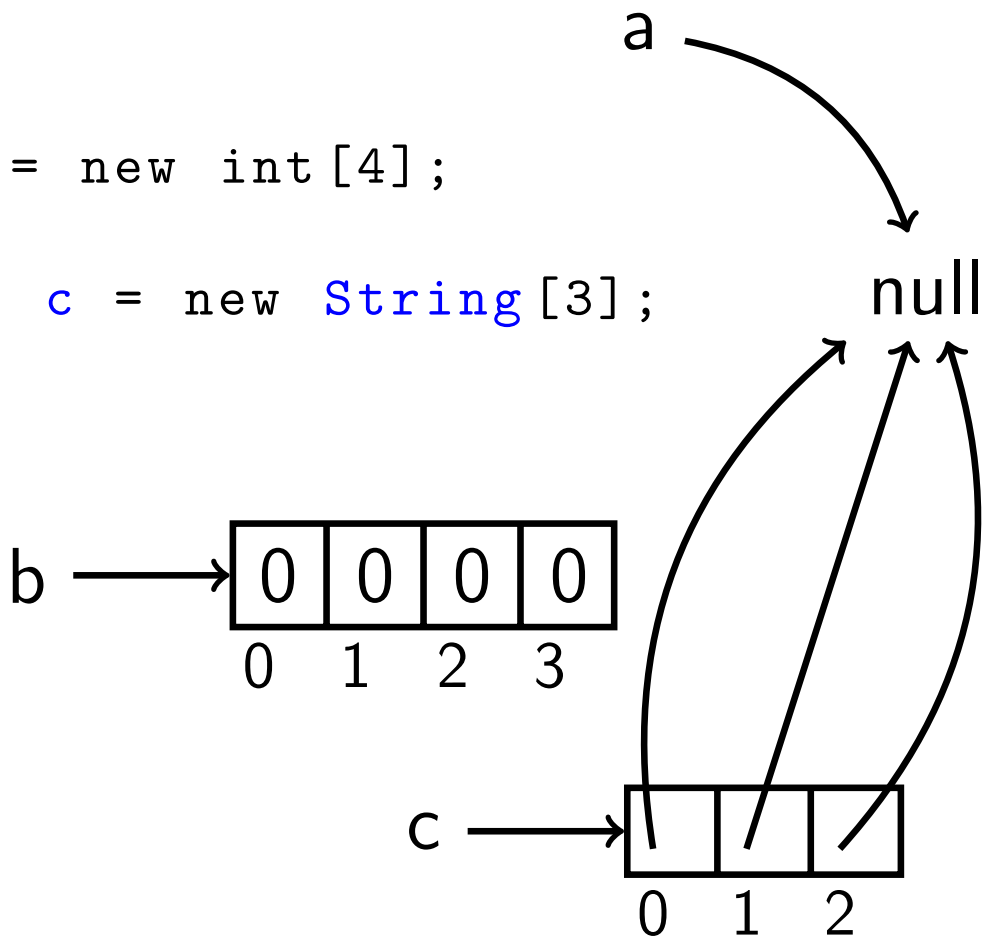
```
int[] a;

int[] b = new int[4];

String[] c = new String[3];
```





Array declaration

```
int[] a;

int[] b = new int[4];

String[] c = new String[3];
```



Accessing array values

- Just like in math, we can read and assign to different indices of our variables.
- In the following example, I'm assuming that indexed variables in math are 1-based, and that appropriate Java arrays (of the right type) have already been created.

Math	**Java**
x_i	`x[i-1]`
$y = x_3$	`y = x[2];`
$x_5 = 15.67$	`x[4] = 15.67;`
$k = \frac{x_1 - x_2}{y_1 - y2}$	`k = (x[0]-x[1])/(y[0]-y[1]);`

Array Initialization

- Arrays can be directly initialized to values by using what's called "Array Initializers":
  - `int[] arr = {5, 3, 8, 4};`
    Creates an int array of length 4 with above values.
  - `String[] sArr = {"Hello", "World"};`
    Creates a String array of length 2 with the above values
- Note! Java arrays are *immutable* once created. This means:
  - You can change values of the elements
  - You can not change the length of the array once it's been created.

Assigning arrays to each other

Since Java arrays are reference types we have to take some special considerations when trying to assign one to another:

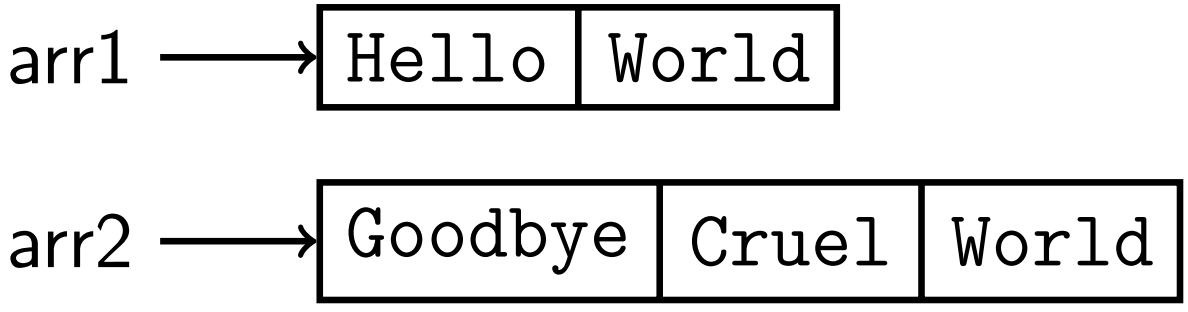
```
String[] arr1 = { "Hello", "World" };
String[] arr2 = { "Goodbye", "World" };
arr1 = arr2; // Array assignment
```

- In the above example both variables `arr1` and `arr2` now refer to the array `["Goodbye", "World"]`, and no variable refers to the original `arr1`.
- When an object (in this case an array) no longer has any variables referring to it, its memory is eventually recycled by means of the "garbage collector".

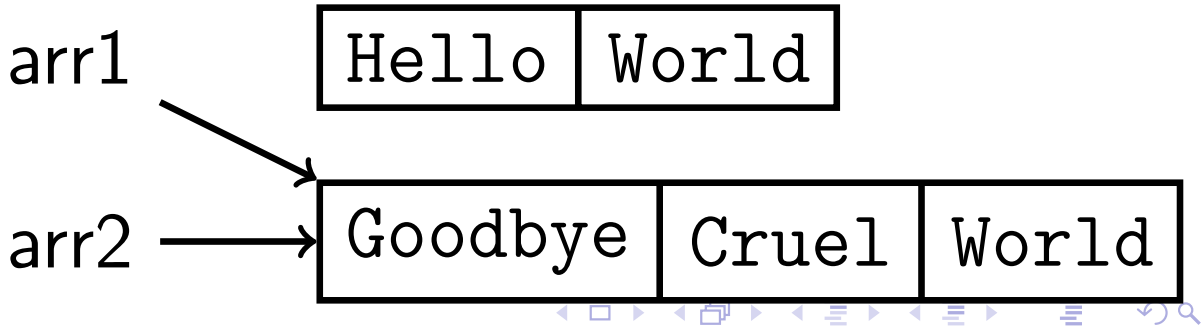
Assigning arrays to each other

```
String[] arr1 = { "Hello", "World" };
String[] arr2 = { "Goodbye", "Cruel", "World" };
arr1 = arr2; // Array assignment
```

Before assignment

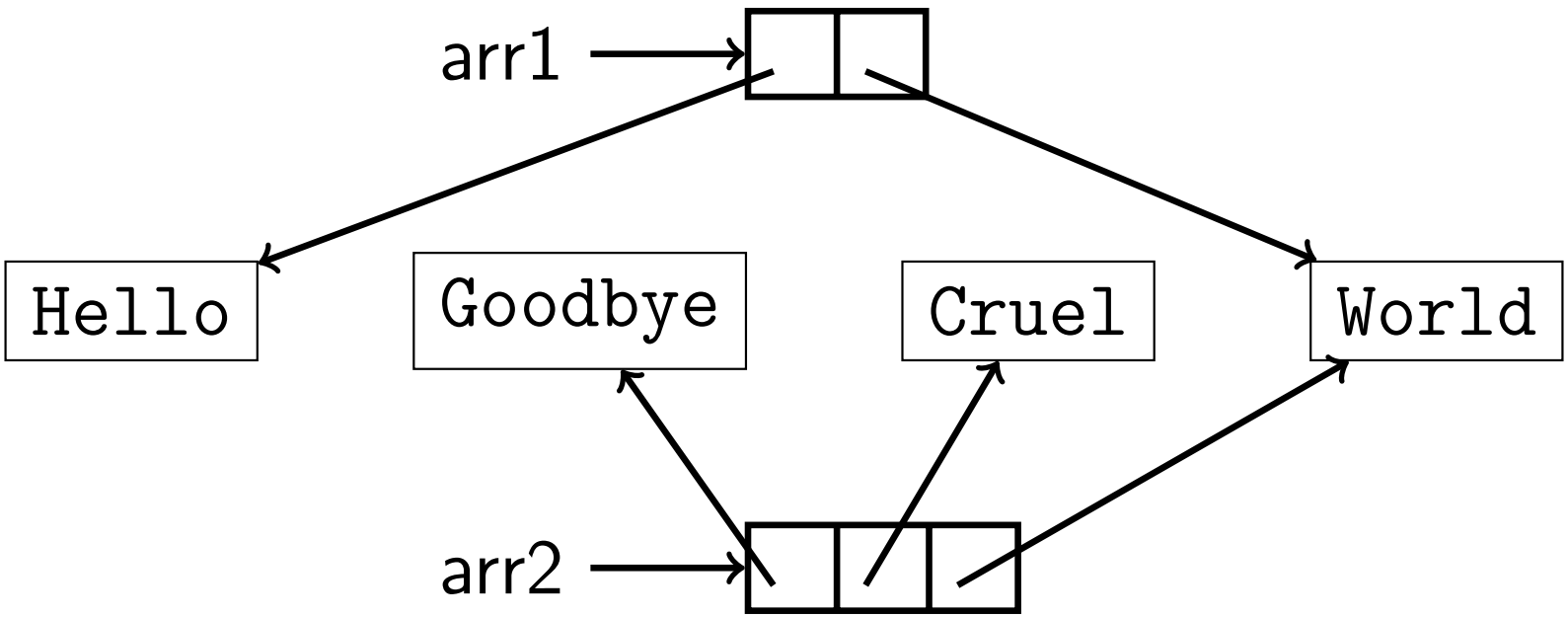


After assignment



Array of Objects

```
String[] arr1 = { "Hello", "World" };
String[] arr2 = { "Goodbye", "Cruel", "World" };
```



Array Example

```
public class ArrayExample1 {
  public static void main ( String[] args ) {
    int[] x = new int[15]; // Array with 15 elements
    int[] y = new int[15];

    // Give each element a value
    for ( int i = 0; i < x.length; i++ ) {
      x[i] = i;
      y[i] = x.length - i - 1;
    }
    // Print out every element in the array
    for ( int element: x ) {
      System.out.println ( element );
    }
    // Copy values from one array to the other
    for ( int i = 0; i < x.length; i++ ) {
      y[i] = x[i];
    }
  }
}
```