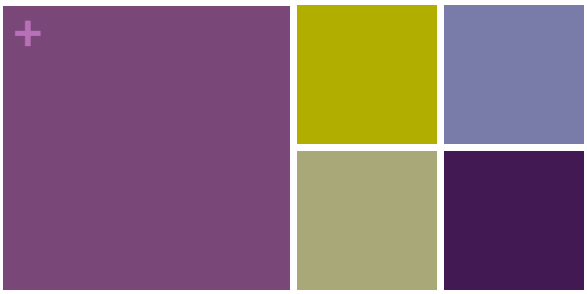

Java vs. C++

Part 2

Language Goals

Java	C++
■ Consistency ■ every file is a class ■ one source file name ■ one object file name ■ one library file name ■ every class is descended from Object ■ Programs are "safe"	■ Specificity ■ every file type has a specific purpose ■ every programming construct has a specific purpose ■ Abstraction at the level you want. ■ Programmer has ultimate control over her program.

Primitive types

Java	C++
■ boolean – true, false ■ not numeric ■ char - 2 bytes (numeric) ■ all numeric types are signed	■ bool – true or false ■ numeric ■ char – 1 byte (numeric integer) ■ numeric types can have unsigned versions ■ integer type modifiers ■ long ■ short ■ signed ■ unsigned

Variables

Java	C++
■ All primitives are values ■ All objects are references ■ There are no pointers	■ all variables are either ■ values ■ int x; ■ references ■ float &y; ■ pointers ■ char* z;

Functions

Java	C++

■ void primitive(int x) {
// the argument passed to
// primitive is copied to
// the stack variable x
}
■ void object(Object y){
// y is a reference to
// the argument passed to
// to the method object
	■ void anyFunction(type x) {
	// the argument passed to
	// anyFunction is copied to
	// the stack variable x
	}
	■ void anyFunction(type &y){
	// y is a reference to
	// the argument passed to
	// to anyFunction

Variables

Java	C++
■ All primitives are values ■ All objects are references ■ There are no pointers	■ all variables are either ■ values ■ int x; ■ references ■ float &y; ■ pointers ■ char* z;

So what is a pointer anyway?

- A pointer is the numeric value that represents the physical address (hopefully) in the program's heap or stack memory.
- For now, the important thing to consider is that pointers have a numeric integer value.

Arrays

Java	C++

int[] x = new int[5];	int x[5];
	for (int i = 0; i < 5; ++i) {
	x[i] = 0;
	}

float[] y = {1.0f,2.0f,3.0f};	float y[] = {1.0,2.0,3.0};
char[] z = {'h','e','l','l','o'};	char [] z = {'h','e','l','l','o'};
	char z2[] = {'h','e','l','l','o','\0'};

Boolean expressions

Java	C++
booleans are NOT NUMERIC! so boolean expressions must evaluate to either true or false	bools are numeric so all numeric expressions can be treated as boolean expressions
	This means that any of the following CAN be used as a boolean expression: a numeric literal, constant, variable, operation, or function a pointer, or a function returning a pointer

Boolean expressions

Java	C++
booleans are NOT NUMERIC! so boolean expressions must evaluate to either true or false	bools are numeric so all numeric expressions can be treated as boolean expressions
	This means that any of the following CAN be used as a boolean expression: a numeric literal, constant, variable, or function a pointer, or a function returning a pointer

However, just because the compiler will let you, doesn't mean you should!

Operators

Java	C++
Operators are primitive functions that cannot be changed.	Operators are, unary, binary, or ternary functions that can be overloaded to work with new types.
Arithmetic operators only work on numeric types.	Arithmetic operators work the same as in java
Boolean operators only work on boolean types	Boolean operators work on numeric types
concatenating Strings uses the + operator.	the + operator works to concatenate std::strings, but not char* (c style) strings.

Non-primitive Types

Java	C++
Everything is a Class in java However, there is an Enum Type class that uses the keyword enum to define ordered constant values with names.	c++ has c types in addition to classes enum (similar to java enum) struct (think of the members part of the class without the functions) union (a way to pack data in multiple formats using the same type) c++ also has classes

Enum

Java

```
public enum ThreatLevel {
   LOW, GUARDED, ELEVATED,
   HIGH, SEVERE }
```

C++

```
enum ThreatLevel {
   LOW, GUARDED=3,
   ELEVATED, HIGH=10,
   SEVERE };

   not strongly typed, just treated
   as integers for strongly typed
   enum use

enum class ThreatLevelStrong {
   LOW, GUARDED=3,
   ELEVATED, HIGH=10,
   SEVERE };
```

Struct (meant to structure data)

Java

```
class MyStruct {
   public int x;
   public int y;

   public static void
main(String[] args) {

    MyStruct js =
        new MyStruct();
    // these have already been
    // initialized to 0
    js.x = 10;
    js.y = 10;
  }
}
```

C++

```
struct MyStruct {
   int x;
   int y;
};


MyStruct cs;
// these must be initialized
cs.x = 10;
cs.y = 10;
```

Classes

Java

```
class MyClass {
   private int x;

   public void setX(int x) {
      this.x = x;
   }

}
```

C++

```
class MyClass {
   int x;

public:
   void setX(int val);

};

void MyClass::setX(int val){
   x = val;
}
```

Strings

Java

```
public class StringTest {
   public static void
     main(String[] args) {
       String x = "hello";
       String y = "world";
       String z = x + ", " + y + "!\n";
       System.out.print(z);
   }
}
```

C++

```
#include <string>
#include <iostream>
using namespace std;
int main() {
  string x = "hello";
  string y = "world";
  string z = x + ", " + y + "!\n";
  cout << z;
}
```

C-Style Strings

- char* x = "hello";

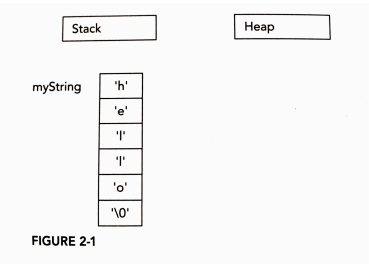

FIGURE 2-1

C-Style Strings

- char* x = "hello";

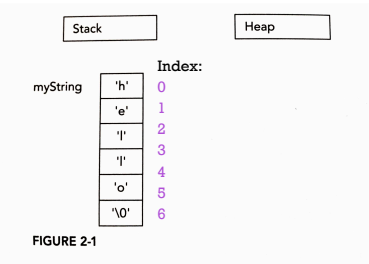

FIGURE 2-1

Streams

Java

- Input
  - System.in – the stream
  - Scanner = new Scanner(System.in);
  - (must use exceptions)
- Output
  - System.out – the stream
  - System.err – the stream

C++

- Input
  - cin – the stream
  - >> – the operator
    - works on all primitive types
    - default delimiter is whitespace
- Output
  - cout – the stream
  - cerr – the stream
  - << – the operator
    - works on all primitive types

cin

- methods (pgs 354-357)
  - fail() – the stream has reached a bad state
  - get() – gets the next character in the stream
    - last char is `std::char_traits<char>::eof()`
  - unget() – puts the last character back on the stream
  - putback(char p) – puts p at the beginning of the stream
  - peek() – shoes what next will give without removing it from the stream
  - getline(char[] buffer, int bufferSize) – takes up to buffersize length line off of the stream and puts it in buffer

cin

- input manipulators (pg. 358)
  - used in conjunction with >> operator
  - boolalpha – reads false as false and anything else as true
  - noboolalpha* – reads 0 as false and anything else as true
  - hex, oct, dec* – specify the numeric base (16, 8, or 10).
  - skipws* – skip over all the whitespace in the stream
  - noskipws – don't skip all of the whitespace in the stream
  - ws – skip the current sequence of whitespace at the current loc.
  - get_money – reads money format
  - get_time – reads time format
  - quoted – reads a string in quotes
  - (* means default)

cout

- output manipulators (pg. 351)
  - most are complementary to cin
  - setprecision – number of decimal places to use
  - setw – the field width of numerical output
  - setfill – the character to pad the output of a number smaller than the value of setw
  - showpoint – show the decimal point of the floating point number
  - noshowpoint – don't show the decimal point of the floating point number with no fractional part.