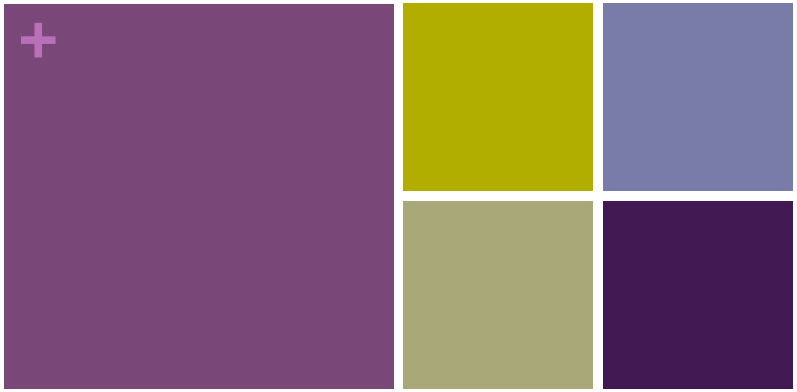

Java vs. C++

Part 1

On the surface: file names

Java	C++
.java – source files	.h – header files
.class – JVM files	.cpp – source files
.jar – library files	.o – object files
	.so – shared library files
	.dylib – shared library files
	.a – static library files
	.dll – shared library files

On the surface: directory structure

Java	C++

src/
  edu/
  brynmawr/
  cs246/
  Introduction.java
	include/
	  introduction.h

class/
  edu/
  brynmawr/
  cs246/
  Introduction.class
	src/
	  introduction.cpp

	bin/ (if it's a program) introduction
	lib/ (if it's part of a program) introduction.o

On the surface: directory structure

Java	C++

src/
  edu/
  brynmawr/
  cs246/
  Introduction.java
	include/
	  introduction.h

class/
  edu/
  brynmawr/
  cs246/
  Introduction.class
	src/
	  introduction.cpp

	bin/ (if it's a program) introduction
	lib/ (if it's part of a program) introduction.o

Required by the JVM based on packages.

On the surface: directory structure

Java	C++
src/ edu/ brynmawr/ cs246/ Introduction.java class/ brynmawr/ cs246/ Introduction.class	include/ introduction.h src/ introduction.cpp bin/ (if it's a program) introduction lib/ (if it's part of a program) introduction.o

Required by the JVM based on packages.

Done by convention.

code structure

Java

```
import java.util.Vector;
public class VectorExample {
    public static void
        main(String[] args) {

        Vector<String> test =
            new Vector<String>();
        for(int i = 0;
            i < args.length; ++i) {
            test.add(args[i]);
        }
        for(String j : test) {
            System.out.println(j);
        }
    }
}
```

C++

```
#include <vector>
#include <iostream>
using namespace std;
int main(int argc, char* argv[]) {

  vector<char*> test;

  for (int i = 0;
       i < argc; ++i) {
    test.insert(test.begin(), argv[i]);
  }
  for (char* j : test) {
    cout << j << endl;
  }
  return 0;
}
```

code structure

Java

```
import java.util.Vector;
public class VectorExample {
    public static void
        main(String[] args) {
        Vector<String> test =
            new Vector<String>();
        for(int i = 0;
            i < args.length; ++i) {
            test.add(args[i]);
        }
        for(String j : test) {
            System.out.println(j);
        }
    }
}
```

import includes package and class definitions

C++

```
#include <vector>
#include <iostream>
using namespace std;
int main(int argc, char* argv[]) {

  vector<char*> test;

  for (int i = 0;
       i < argc; ++i) {
    test.insert(test.begin(), argv[i]);
  }
  for (char* j : test) {
    cout << j << endl;
  }
  return 0;
}
```

code structure

Java

```
import java.util.Vector;
public class VectorExample {
    public static void
        main(String[] args) {
        Vector<String> test =
            new Vector<String>();
        for(int i = 0;
            i < args.length; ++i) {
            test.add(args[i]);
        }
        for(String j : test) {
            System.out.println(j);
        }
    }
}
```

import gets class definitions and implied package use

C++

```
#include <vector>
#include <iostream>
using namespace std;
int main(int argc, char* argv[]) {

  vector<char*> test;

  for (int i = 0;
       i < argc; ++i) {
    test.insert(test.begin(), argv[i]);
  }
  for (char* j : test) {
    cout << j << endl;
  }
  return 0;
}
```

#include gets only definitions

code structure

Java

```
import java.util.Vector;
public class VectorExample {
    public static void
        main(String[] args) {
        Vector<String> test =
            new Vector<String>();
        for(int i = 0;
            i < args.length; ++i) {
            test.add(args[i]);
        }
        for(String j : test) {
            System.out.println(j);
        }
    }
}
```

C++

```
#include <vector>
#include <iostream>
using namespace std;
int main(int argc, char* argv[]) {

  vector<char*> test;

  for (int i = 0;
       i < argc; ++i) {
    test.insert(test.begin(), argv[i]);
  }
  for (char* j : test) {
    cout << j << endl;
  }
  return 0;
}
```

import gets class definitions and implied package use

using namespace gets implied namespace use

code structure

what else is different?

Java

```
import java.util.Vector;
public class VectorExample {
    public static void
        main(String[] args) {

        Vector<String> test =
            new Vector<String>();
        for(int i = 0;
            i < args.length; ++i) {
            test.add(args[i]);
        }
        for(String j : test) {
            System.out.println(j);
        }
    }
}
```

C++

```
#include <vector>
#include <iostream>
using namespace std;
int main(int argc, char* argv[]) {

  vector<char*> test;

  for (int i = 0;
       i < argc; ++i) {
    test.insert(test.begin(), argv[i]);
  }
  for (char* j : test) {
    cout << j << endl;
  }
  return 0;
}
```

compiling

Java

- **javac** [options] [sourcefiles] [@argfiles]

C++

```
g++ [-c|-S|-E] [-std=standard]
    [-g] [-pg] [-Olevel]
    [-Wwarn...] [-Wpedantic]
    [-Idir...] [-Ldir...]
    [-Dmacro[=defn]...]
    [-Umacro]
    [-foption...]
    [-mmachine-option...]
    [-o outfile] [@file]
    infile...
```

Only the most useful options are listed here; see man page for the remainder.

compiling

Java

- **javac** [options] [sourcefiles] [@argfiles]
- Compiling an application:
- javac <file1.java> …
- example:
  - java HelloWorld.java

creates file:
HelloWorld.class

C++

```
g++ [-c] [-o outfile]
    infile...
```

We'll start with these.

Compiling an application:

g++ <file1.cpp> …

example:

g++ hello.cpp

creates file:
a.out

compiling

Java	C++
■ **javac** [options] [sourcefiles] [@argfiles]	g++ [-c] [-o outfile] infile...
■ Compiling an application:	We'll start with these.
■ javac <file1.java> …	Compiling an application:
■ example:	g++ -o <appname> <file1.cpp> …
■ java HelloWorld.java	example:
creates file: HelloWorld.class	g++ -o HelloWorld hello.cpp
	creates file: HelloWorld

compiling

Java	C++
■ **javac** [options] [sourcefiles] [@argfiles]	g++ [-c] [-o outfile] infile...
■ Compiling a part of an application:	We'll start with these.
■ javac <file1.java> …	Compiling a part of an application:
■ example:	g++ -c -o <objname.o> <file1.cpp> …
■ java Hello.java	example:
creates file: Hello.class	g++ -c -o hello.o hello.cpp
	creates file: hello.o

running

Java	C++
■ java <className>	■ ./<appname>
■ Example:	■ Example
■ java HelloWorld	■ ./HelloWorld
java edu.brynmawr.cs246.Introduction	■ Or, if the program is in your path: <appname>
	■ Example:
	■ ls
	■ cd
	■ cat
	■ emacs

Language Goals

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

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