Java Resources

Java for C++ Programmers – **MUST READ**
http://triton.towson.edu/~mzimand/os/Lect2-java-tutorial.html

Learning a New Programming Language: Java for C++ Programmers
http://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial

Sun Java Tutorials
http://java.sun.com/docs/books/tutorial/

Java 1.5 API Reference
http://java.sun.com/j2se/1.5.0/docs/api/index.html

What's different?

- No delete
- No pointers
- OOP "The way it should be"
- Platform independent

```
class Pair{int x,y;}

class PairExample
{
   void f() {
   int n = 1;
   Pair p = new Pair();          // this is the way to initialize p
   p.x = 2; p.y = 3;
   System.out.println(n);        // prints 1
   System.out.println(p.x);      // prints 2
   g(n,p);
   System.out.println(n);        // still prints 1
   System.out.println(p.x);      // prints 100
   }

void g(int num, Pair ptr) {
   System.out.println(num);      // prints 1
   num = 17;                     // changes only the local copy
   System.out.println(num);      // prints 17
   System.out.println(ptr.x);    // prints 2
   ptr.x = 100;                  // changes x field of caller's Pair
   ptr = null;                   // changes only the local ptr
   }
}
```

What you should know…

No stand-alone functions

Method definitions are contained within the class definition

main() must

- be inside a class
- be `public static void`
- have one argument, a string array

What you should know…

To write to standard output,

```
System.out.println("something")
System.out.print("something else")
```

Input Output

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
BufferedReader in = new BufferedReader(new FileReader("name_of_file")));
```

Java Naming Conventions

classes	class Account class BankAccount
methods	void turnRight() int startLeftMotor()
variables	int headCount
constants	static final int MAX_VALUE = 10

Static, Final and Public…

Static

– only one member shared across all instances

Final

– constant (similar to const)

Public,Private,Protected

– Similar to C++

Inheritance

Extends
– Derived class inherits methods of parent class
– Can override members of parent class
– Only ONE parent

Implements
– `interface` … {} describes public members of class
– a class that implements an interface must
define those members
– one class can implement multiple interfaces

Example of inheritance

```
class BaseClass{
   int inheritableMember;
}
interface Move{
   void goStraight();
   void turnLeft();
}
class DerivedClass extends BaseClass implements Move
{
   void goStraight(){
   ...                  // mandatory implementations of
   }                    //functions
   void turnLeft()
   {...}                // these may use inheritableMember
}
```

Exception Handling

```
try {
      ...
      foo.bar();
      ...
      a[i] = 17;
      ...
    }catch (IndexOutOfBoundsException e)
      {System.err.println(“oops: " + e);}
```

Word Tokenizer Example

We will build a complete program from scratch

Demonstrates reading from input

The `StreamTokenizer` class it uses may come in handy for your project

```
import java.io.*;
/* import is like #include, except it's not a literal copy and
   paste. Allows the use of other classes in other files.
   java.io is a special class that comes with the language (much like
   <iostream>) */


class WordLocatorTokenizer {
// Members (variables and functions) of class go here
// Functions are defined inside class definition. (unlike C++)
}
```

```
import java.io.*;

class WordLocatorTokenizer {
   public static void main(String[] args) {
        // Main definition goes here
   }
   /*
public - means same thing as in C++
static - same as in C++ (one instance across all instances of class)
void - same as in C++
main() - called when WordLocatorTokenizer is started as a program
String[] args - like *argv[] in c++ main definition. argc is not needed
because string objects have a .length attribute.
   */
}
```

```
import java.io.*;

class WordLocatorTokenizer {
   public static void main(String[] args) {
        BufferedReader in =
                new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer st = new StreamTokenizer(in);
// ...
   }
}
```

```
import java.io.*;

class WordLocatorTokenizer {
   public static void main(String[] args) {
        BufferedReader in =
                new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer st = new StreamTokenizer(in);

        while (true) {
                int nexttoken = st.nextToken();

                if (nexttoken == StreamTokenizer.TT_WORD) {
                        System.out.println("Word: " + st.sval);

                } else if (nexttoken == StreamTokenizer.TT_NUMBER) {
                        System.out.println("Number: " + st.nval);

                } else if (nexttoken == StreamTokenizer.TT_EOF) {
                        System.out.println("All tokens Read");
                        return;
                }
        }
   }
}
```

Compiling...

```
$ javac WordLocatorTokenizer.java
WordLocatorTokenizer.java:10: unreported exception
java.io.IOException; must be caught or declared to be
thrown
int nexttoken = st.nextToken();
^
1 error
$
```

```
import java.io.*;

class WordLocatorTokenizer {
   public static void main(String[] args) {
        BufferedReader in =
                new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer st = new StreamTokenizer(in);
        try {
           while (true) {
                   int nexttoken = st.nextToken();
                   if (nexttoken == StreamTokenizer.TT_WORD) {
                        System.out.println(“word: ” + st.sval);

                   } else if (nexttoken == StreamTokenizer.TT_NUMBER) {
                        System.out.println(“Number: ” + st.nval);

                   } else if (nexttoken == StreamTokenizer.TT_EOF) {
                        System.out.println("All tokens Read");
                        return;
                   }
           }

        } catch (java.io.IOException e) {
                System.out.println(".... End ...");
        }
   }
}
```

What will happen on compiling?

```
class ClassName
{
   void functionCall()
   {
        ...
   }

   public static void main(String[] args)
   {
        ...
        functionCall();
        ...
   }
}
```

Some things to go over…

- String methods
- Arrays
- Collections
- List
- Vector
- Map