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,

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

Input Output

```java
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
BufferedReader in = new BufferedReader(new FileReader("name_of_file"));
```
# Java Naming Conventions

| classes      | class Account  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>class BankAccount</td>
</tr>
</tbody>
</table>
| 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
Packages

• Similar in function to namespaces in C++

• Organizes code into hierarchies similar to directories

• Structure of packages mirrors structure on disk

• You've already seen this structure while using import!
  import java.io.*;
Packages

• Every class is part of some package.

• All classes in a file are part of the same package.

• Packages are specified on the first (non-comment) line in the file.
  package name;

• Multiple files can specify the same package name.

• If no package is specified, the classes in the file go into a special unnamed (default) package

• If package <name> is specified, the file must be in a subdirectory called <name> (i.e., the directory name must match the package name).
Using Packages

```java
class ListNode { /*...*/ }

package ListPkg;
public class List { /*...*/ } 

In the file that uses List,
ListPkg.List varName = new ListPkg.List();

Instead,
import ListPkg; 
... 
... 
List varName = new List();
```
Strings

String s = "hello";
String t = "world";

System.out.println(s + " , " + t);   // prints "hello, world"
System.out.println(s + "1234");     // "hello1234"
System.out.println(s + (12*100 + 34)); // "hello1234"
System.out.println(s + 12*100 + 34);  // "hello120034" (why?)
System.out.println("The value of x is " + x); // will work for any x

String numbers = "";
for (int i=0; i<5; i++)
    numbers += " " + i;
System.out.println(numbers); // " 0 1 2 3 4"
Strings

• Every class is a descendant of Object
• All classes inherit ancestor's methods
• Object has a toString() method.
• Therefore...

• Override your classes' toString() methods!!
Strings

There are many useful string methods
• substring()
• compareTo()
• charAt()
• length()
• toLowerCase()
• matches()
• split()
Regular Expressions

- To match ‘eecs’ $\rightarrow$ “eecs”
- To match gray/grey $\rightarrow$ “gr[ea]y”
- To match digit $\rightarrow$ “[0-9]”
- To match anything except digits $\rightarrow$ “[^0-9]”
- To match any character $\rightarrow$ “.”
- To match any 3 digits $\rightarrow$ “[0-9]{3}”
- To match any no.($\geq 0$) of digits $\rightarrow$ “[0-9]∗”
- To match at least 1 digit $\rightarrow$ “[0-9]+”
Regular Expressions

Identify

• Any hexadecimal?
• Variable name starting with alphabet?
• Number between 100 and 99999?
• Any character other than letters?
Collection<E>

- Similar to C++ STL
  http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html

- New versions allow for java generics

- Generics are like C++ templates

- Prior to generics, use of Collections required nasty typecasting
Example

• Our example uses ArrayList\<E\>
  Just like vector
  Java also has a vector\<\> class for legacy reasons.
  It works about the same, but has different properties in
  multi-thread programs

• And there are many more collection types!
  Set
  Queue
  Map

• Many have multiple implementation options

• LinkedList has same interface as ArrayList, but different runtime
  properties
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListDemo {
    public static void main(String a[]) {
        List<String> v = new ArrayList<String>(); // List <-> ArrayList
        v.add( "EECS" );
        v.add( "484" );
        v.add( "Is Awesome!" );
        System.out.println( v ); // uses ArrayList.toString()

        for( int i = 0; i < v.size(); i++) { // C++ style array iteration
            String s = v.get(i);
            System.out.print( s + " ");
        }

        Iterator<String> itr = v.iterator();
        while (itr.hasNext()) { // Iterator Style Iteration
            String s = itr.next();
            System.out.print(s + " ");
        }

        for(String s: v){ // Java 1.5 for loop
            System.out.print( s + " ");
        }
    }
}
Generics

• Communicates the type of a collection to the compiler

• Compiler checks if collection used consistently
Generics

// Removes 4-letter words from c. Elements must be strings
static void expurgate(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); )
        if (((String) i.next()).length() == 4)
            i.remove();
}

// Removes the 4-letter words from c
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator(); i.hasNext(); )
        if (i.next().length() == 4)
            i.remove();
}