Today

- Packages
- Strings
- Regular Expressions
- Collection<E>
- Generics/Casts
- Project Questions

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

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

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’ → “eecs”
- To match gray/grey → “gr[ea]y”
- To match digit → “[0-9]”
- To match anything except digits → “[^0-9]”
- To match any character → “.”
- To match any 3 digits → “[0-9]{3}”
- To match any no.(>=0) of digits → “[0-9]*”
- To match atleast 1 digit → “[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();
}
```