```java
import java.util.Arrays;
import java.util.List;

public abstract class Env {

    /**
     * Look up the symbol in the environment.
     */
    public abstract Val applyEnv(String sym);

    /**
     * Extend the current environment by adding bindings.
     * @return a new Env linked to this one.
     */
    public Env extendEnv(Bindings bindings) {
        return new EnvNode(bindings, this);
    }

    /**
     * Display, in a string, the sequence of all bindings,
     * newest to oldest, in this and in enclosing environments.
     */
    @Override
    public String toString() {
        return "{" + this.envListing() + "}";
    }

    /**
     * Show the bindings in this Env. and in all surrounding Env's.
     */
    public abstract String envListing();

    /**
     * The "end of sequence" object for an environment chain
     */
    private static class EnvNull extends Env {
        /**
         * The parent class will make one instance of this class.
         */
        private EnvNull () {}

        @Override
        public Val applyEnv(String sym) {
            throw new RuntimeException("no binding for "+sym);
        }

        @Override
        public String envListing() {
            return "";
        }
    }

    /**
     * The initial (empty) environment
     */
    private static final Env ENV_NULL = new EnvNull();
```



```java
    public static void main(String [] args) {
        Env env0 = Env.ENV_NULL;
        Env env1 = env0.extendEnv(
            new Bindings(Arrays.asList(
                new Binding("a", new Val(1)),
                new Binding("b", new Val(2)),
                new Binding("c", new Val(3)))));
        List<String> i2 = Arrays.asList("a", "p");
        List<Val> v2 = Arrays.asList(new Val(4), new Val(5));
        Env env2 = env1.extendEnv(new Bindings(i2, v2));
        try {
            System.out.println("env0:\n" + env0);
            System.out.println("env1:\n" + env1);
            System.out.println("env2:\n" + env2);
            System.out.print("a(env2) => ");
                System.out.println(env2.applyEnv("a"));
            System.out.print("a(env1) => ");
                System.out.println(env1.applyEnv("a"));
            System.out.print("b(env2) => ");
                System.out.println(env2.applyEnv("b"));
            System.out.print("p(env2) => ");
                System.out.println(env2.applyEnv("p"));
            System.out.print("p(env1) => ");
                System.out.println(env1.applyEnv("p"));
        }
        catch( Exception e ) {
            System.err.println( e );
        }
    }
}
```

```java
/**
 * The standard Env class. It contains a Bindings object.
 */
public class EnvNode extends Env {

    /** Sequence of local bindings */
    public final Bindings bindings;

    /** Enclosing scope in the chain */
    public final Env enclosing;

    /**
     * Create a new environment
     * @param bindings a given set of variable bindings
     * @param env the enclosing scope
     */
    public EnvNode(Bindings bindings, Env env) {
        this.bindings = bindings;
        this.enclosing = env;
    }

    /**
     * Look up the given symbol in this Env's bindings.
     * @param sym the symbol
     * @return the symbol's (most recent) binding in the Env chain
     */
    @Override
    public Val applyEnv(String sym) {
        // look first in the local bindings
        for (Binding b : bindings.bindingList) {
            if (sym.equals(b.id))
                return b.value;
        }
        // not found in the local bindings,
        // so look in the next (enclosing) environment
        return enclosing.applyEnv(sym);
    }

    /**
     * Create a string containing the bindings of this environment,
     * plus those of enclosing environments.
     */
    @Override
    public String envListing() {
        String result = bindings.toString();
        result += enclosing.envListing();
        return result;
    }
}
```

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * A set of bindings associated with a single scope.
 * A sequence of bindings forms an environment for the program at any
 * given point in time.
 */
public class Bindings {

    /**
     * An associative table of identifier bindings
     */
    public final List< Binding > bindingList;

    /**
     * Bindings is initially empty, unless another constructor is used.
     */
    public Bindings() {
        bindingList = new ArrayList< Binding >();
    }

    /**
     * Create bindings from the parallel arrays constructed by code
     * generated by PLCC.
     */
    public Bindings( List< String > idList, List< Val > valList ) {
        // the Lists must be the same size
        if ( idList.size() != valList.size() )
            throw new RuntimeException( "Bindings: List size mismatch" );
        bindingList = new ArrayList< Binding >();
        Iterator< String > is = idList.iterator();
        Iterator< Val > vs = valList.iterator();
        while ( is.hasNext() ) {
            bindingList.add( new Binding( is.next().toString(), vs.next() ) );
        }
    }

    /**
     * Create Bindings from a list of Bindings.
     */
    public Bindings( List< Binding > bindingList ) {
        this.bindingList = bindingList;
    }

    /**
     * Add a Binding object to this local environment.
     */
    public void add( Binding b ) {
        bindingList.add( b );
    }

    /**
     * Add a binding (s, v) to this local environment.
     */
```



```java
    public void add( String s, Val v ) {
        add( new Binding( s, v ) );
    }

    /**
     * Create a string that is the concatentated string representations
     * of the individual Bindings.
     */
    @Override
    public String toString() {
        String result = "";
        for ( Binding b : bindingList ) {
            result += b;
        }
        return result;
    }
}
```

Jan 26 22:15 2020 Binding.java Page 1

```java
/**
 * Binding of a single identifier to its value.
 * They go into a collection of bindings called Bindings.
 */
public class Binding {
    /** the identifier / variable name */
    public final String id;

    /** the value to which the identifier is bound */
    public final Val value;

    /**
     * Construct a Binding by initializing its two fields.
     */
    public Binding( String id, Val value ) {
        this.id = id;
        this.value = value;
    }

    /**
     * Return a string representation in the format '(ID,value)'.
     */
    @Override
    public String toString() {
        return "(" + id + ',' + value + ')';
    }
}
```

Jan 26 22:14 2020 Val.java Page 1

```java
/**
 * The run-time value of a variable
 */
public class Val {

    /**
     * The actual int value
     */
    public final int value;

    /**
     * Create an int Val.
     */
    public Val( int value ) {
        this.value = value;
    }

    /**
     * Return the int value as a string.
     */
    @Override
    public String toString() {
        return Integer.toString( value );
    }
}
```