Appendix A

Refactoring Examples

Convert Local Variable to Field

Simple Example

Before:

```
public class MyClass {
    public void myMethod() {
        int myLocalVariable = 10;
        System.out.println(myLocalVariable);
    }
}
```

After:

```
public class MyClass {
    private int myField; // Converted from local variable

    public void myMethod() {
        myField = 10;
        System.out.println(myField);
    }
}
```

Real Example

Before:

```
public class BasicDynaBean implements DynaBean, Serializable {
...
 protected DynaProperty getDynaProperty(String name) {

        DynaProperty descriptor = getDynaClass().getDynaProperty(name);
        if (descriptor == null) {
            throw new IllegalArgumentException
                    ("Invalid property name '" + name + "'");
        }
        return (descriptor);

    }
}
```

After:

```
public class BasicDynaBean implements DynaBean, Serializable {
    public DynaProperty descriptor; // Converted from local variable
...
    protected DynaProperty getDynaProperty(String name) {

        descriptor = getDynaClass().getDynaProperty(name);
        if (descriptor == null) {
            throw new IllegalArgumentException
                    ("Invalid property name '" + name + "'");
        }
```

```
        return (descriptor);

    }
}
```

Extract Local Variable

Simple Example

Before:

```
public class MyClass {
    public void myMethod() {
        int result = 2 * (3 + 5);
        System.out.println(result);
    }
}
```

After:

```
public class MyClass {
    public void myMethod() {
        int sum = 3 + 5; // Extracted local variable
        int result = 2 * sum;
        System.out.println(result);
    }
}
```

Real Example

Before:

```
File rootFile = rootEntry.getFile();
if (rootFile.exists()) {
    checkAndNotify(rootEntry, rootEntry.getChildren(), listFiles(rootFile));
} else if (rootEntry.isExists()) {
    checkAndNotify(rootEntry, rootEntry.getChildren(), FileUtils.EMPTY_FILE_ARRAY);
}
```

After:

```
File rootFile = rootEntry.getFile();
FileEntry[] children = rootEntry.getChildren(); // Extracted local variable
if (rootFile.exists()) {
    checkAndNotify(rootEntry, children, listFiles(rootFile));
} else if (rootEntry.isExists()) {
    checkAndNotify(rootEntry, children, FileUtils.EMPTY_FILE_ARRAY);
}
```

Extract Method

Simple Example

Before:

```
public class MyClass {
    public void myMethod() {
        int result = 2 * (3 + 5);
        System.out.println(result);
    }
}
```

After:

```
public class MyClass {
    public void myMethod() {
        int result = calculateResult(); // Extracted method
        System.out.println(result);
    }
```

```
    private int calculateResult() {
        return 2 * (3 + 5);
    }
}
```

Real Example

Before:

```
public boolean addAll(Collection coll) {
    boolean changed = false;
    Iterator i = coll.iterator();
    while (i.hasNext()) {
        boolean added = add(i.next());
        changed = changed || added;
    }
    return changed;
}
```

After:

```
public boolean addAll(Collection coll) {
    boolean changed = false;
    changed = extractMethod(coll, changed);
    return changed;
}

private boolean extractMethod(Collection coll, boolean changed) {
    Iterator i = coll.iterator();
    while (i.hasNext()) {
        boolean added = add(i.next());
        changed = changed || added;
    }
    return changed;
}
```

Introduce Indirection

Simple Example

Before:

```
public class MyClass {
    public void myMethod() {
        originalMethod();
    }

    public void originalMethod() {
        System.out.println("Original method");
    }
}
```

After:

```
public class MyClass {
    public void myMethod() {
        indirectMethod(); // Introduced indirection
    }

    public static void indirectMethod() {
        MyClass myClass = new MyClass();
        myClass.originalMethod();
    }

    public void originalMethod() {
        System.out.println("Original method");
    }
}
```

Real Example

```
public abstract class AbstractMapBag implements Bag {
    private transient Map map;
    ...
    public int getCount(Object object) { // Original getCount method
        MutableInteger count = (MutableInteger) map.get(object);
        if (count != null) {
            return count.value;
        }
        return 0;
    }

    boolean containsAll(Bag other) {
        boolean result = true;
        Iterator it = other.uniqueSet().iterator();
        while (it.hasNext()) {
            Object current = it.next();
            boolean contains = getCount(current) >= other.getCount(current); // Original line
            result = result && contains;
        }
        return result;
    }
    ...
}
```

After:

```
public abstract class AbstractMapBag implements Bag {
    public static int getCount(Bag bag, Object object) { // Added static method
        return bag.getCount(object);
    }

    private transient Map map;
    ...
    public int getCount(Object object) { // Unchanged getCount method
        MutableInteger count = (MutableInteger) map.get(object);
        if (count != null) {
            return count.value;
        }
        return 0;
    }

    boolean containsAll(Bag other) {
        boolean result = true;
        Iterator it = other.uniqueSet().iterator();
        while (it.hasNext()) {
            Object current = it.next();
            boolean contains = AbstractMapBag.getCount(this, current) >= AbstractMapBag.getCount(other,
                current); // Changed line to use static method
            result = result && contains;
        }
        return result;
    }
    ...
}
```

Inline Method

Simple Example

Before:

```
public class MyClass {
    public void myMethod() {
        printMessage();
    }

    public void printMessage() {
        System.out.println("Hello, World!");
    }
}
```

After:

```
public class MyClass {
    public void myMethod() {
        System.out.println("Hello, World!"); // Inlined method
    }
}
```

Real Example

Before:

```
protected void removeMapping(HashEntry entry, int hashIndex, HashEntry previous) {
    modCount++;
    removeEntry(entry, hashIndex, previous);
    size--;
    destroyEntry(entry);
}

protected void destroyEntry(HashEntry entry) {
    entry.next = null;
    entry.key = null;
    entry.value = null;
}
```

After:

```
protected void removeMapping(HashEntry entry, int hashIndex, HashEntry previous) {
    modCount++;
    removeEntry(entry, hashIndex, previous);
    size--;
    entry.next = null; // inlined method
    entry.key = null;
    entry.value = null;
}
```

Introduce Parameter Object

Simple Example

Before:

```
public class MyClass {
    public void myMethod(String firstName, String lastName) {
        System.out.println(firstName + " " + lastName);
    }
}
```

After:

```
public class MyClass {
    public void myMethod(Person person) { // Introduced parameter object
        System.out.println(person.getFirstName() + " " + person.getLastName());
    }
}

class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
```

```
    }
}
```

Real Example

Before:

```
public abstract class AbstractRealDistribution implements RealDistribution, Serializable {
    double x = UnivariateSolverUtils.solve(toSolve,lowerBound,upperBound,getSolverAbsoluteAccuracy());
...
}
public class UnivariateSolverUtils {
    public static double solve(UnivariateFunction function, double x0,
                                                             double x1,
                                                             double absoluteAccuracy) {
    if (function == null) {
        throw new NullArgumentException(LocalizedFormats.FUNCTION);
    }
    final UnivariateSolver solver = new BrentSolver(absoluteAccuracy);
    return solver.solve(Integer.MAX_VALUE, function, x0, x1);
    }
...
}
```

After:

```
public class SolveParameter {
    public UnivariateFunction function;
    public double x0;
    public double x1;
    public double absoluteAccuracy;

    public SolveParameter(UnivariateFunction function, double x0, double x1, double absoluteAccuracy) {
    this.function = function;
    this.x0 = x0;
    this.x1 = x1;
    this.absoluteAccuracy = absoluteAccuracy;
  }
}

public abstract class AbstractRealDistribution implements RealDistribution, Serializable {
    double x = UnivariateSolverUtils.solve(new SolveParameter(toSolve, lowerBound, upperBound,
         getSolverAbsoluteAccuracy()));
...
}

public class UnivariateSolverUtils {
    public static double solve(SolveParameter parameterObject) {
        if (parameterObject.function == null) {
            throw new NullArgumentException(LocalizedFormats.FUNCTION);
        }
        final UnivariateSolver solver = new BrentSolver(parameterObject.absoluteAccuracy);
        return solver.solve(Integer.MAX_VALUE, parameterObject.function, parameterObject.x0,
             parameterObject.x1);
    }
...
}
```