实验二：基于 JoeQ 的程序优化

在本次实验中，同学们将基于上一次实验的数据流分析，进一步完成 JoeQ 程序的机器无关优化。

程序文件与运行方式

目录结构

以下结构出现在你的 `~/exp2` 中：

```
lib/joeq.jar    packaged JoeQ src/examples Examples of using JoeQ
src/flow        The dataflow framework and interfaces, with Solver implemented
src/submit      The code you will submit (MODIFY THIS)
src/test        Tests for your solutions (you may add more tests here)
                *.basic.out includes expected output for FindRedundantNullChecks.
build.xml       Apache Ant build script for building your code
exp2.properties Apache Ant build configurations
run.sh          Executor script for running your code
```

运行方式

本次实验中稍微调整了项目的处理方式，使用了 Apache Ant 进行项目的生成。具体来说，直接在 `~/exp2` 中运行 `ant` 即可完成生成，而后使用 `run.sh` 进行执行。`run.sh` 的第一个参数是主类名，后续参数是 Java 程序运行的参数；事实上它只设置了 `CLASSPATH` 为 `build:lib/joeq.jar` 来省去你指定依赖，其余行为和 `java` 可执行文件本身是完全一致的。

举例来说：

```
$ cd exp2
$ ant
Buildfile: /home/huanqi/exp2/build.xml
( ... )
BUILD SUCCESSFUL
Total time: 0 seconds
$ ./run.sh examples.PrintQuads test.NullTest
Class: test.NullTest
Control flow graph for test.NullTest.getInteger (Z)Ljava/lang/Integer;:
( ... )
$ ./run.sh submit.FindRedundantNullChecks test.NullTest
getInteger
Test1
Test2
getInteger
Test3 39 40
main
<init>
```

本地开发的方式

Intellij IDEA 的 Project from Existing Source 能够很好地工作，不过需要选定 1.5 版本的 JDK；Eclipse 本身具备 Java Project from Existing Ant Build File 这一功能。

为了执行或调试程序，你可以在你的 IDE 中自行添加执行配置，设置主类和参数。

实验内容

1. 实现 `submit.FindRedundantNullChecks`，其功能应为分析并打印输入的所有类名所对应类的各方法的冗余 `NULL_CHECK` Quads。以输入类名 `"test.NullTest"` 为例，其输出应如下：

   ```
   getInteger
   Test1
   Test2
   getInteger
   Test3 39 40
   main
   <init>
   ```

   其中，某行方法名后为空表示该方法没有冗余的 `NULL_CHECK`，非空则表示这些 Quads 是冗余的 `NULL_CHECK`。

   `NULL_CHECK` 这一 Operator 是用于检查一个 Object 是否为 `null` 的。其 Used Register 仅有一个，是被检查的对象寄存器；当为 `null` 时，抛出异常，否则继续执行。它没有输出，尽管被标记为了 `T-1`。

   所谓“冗余”是说，对于某些 `NULL_CHECK` 指令，执行到此处时总是已经被检查过其目标是否为 `null`，因而这一次检查一定不会失败抛出异常，从而这一 Quad 是无意义的、可以直接去除。

   除此之外，作为**额外的加分项**，同学们可以尝试解决与 `null` 进行 `IFCMP_A` 后的 `NULL_CHECK`。在这样的比较确认为 `false` 后，我们已知它不会是 `null`，因而可以消除这样的 `NULL_CHECK`。这可能需要略微调整数据流分析的算法，所需的更改请不要在 `src/flow` 中直接进行，而是在 `src/submit` 中添加新的调整后的实现。仅在输入参数列表包含 `-e` 时进行。

2. 实现 `submit.Optimize.optimize(List<String> optimizeClasses, boolean nullCheckOnly)`。其功能应为对根据类名 `Helper.load` 得到的 `clazz` 进行优化；在 `nullChecksOnly` 为 `true` 时，仅移除冗余的 `NULL_CHECK`，否则也进行其它优化。

   注意，`nullCheckOnly == true` 时，作为基础的考察情况，去除的冗余 `NULL_CHECK` 仅包含先前提及的简单情形，对涉及分支的不要做处理。涉及分支版本的去除冗余优化不算做其他优化。

   只有消除简单冗余 `NULL_CHECK` 是必须完成的任务；其它优化的加入是可选的，计作单独的加分项。每位同学最多完成**两项**（计分的）其它优化，更多的不予加分。你需要在 `src/submit/design.md` 中对你的所有其它优化进行说明。

   这一类中进行了适当的包装，可以作为主类使用，用以测试优化实现。具体来说，

   - 直接指定的参数是要处理的类名。
   - `-e/--extra` 表示执行消除冗余 `NULL_CHECK` 之外，额外的加分优化。
   - `-m/--main` 后的参数是主类名。指定时会解释执行这一主类的 `Main` 方法。
   - `-p/--param` 后的参数是“,”分隔的参数列表，在提供了主类名时作为输入主类的参数。无参数时不提供 `-p` 即可。
   - `--print` 表示输出优化后的 JoeQ 程序。

同学们可以在 `~/src/submit` 中添加新的 `.java` 源文件和相应的类来进行具体的优化实现。Ant 能够对它们进行统一的编译，不需要额外的处理。

Hints

1. 合理利用 `flow.FlowSolver`。这实际是实验一中 MySolver 的样例实现，你可以同样地使用它来进行数据流分析。使用 `Helper.runPass(clazz, solver)` 来对类 `clazz` 的所有方法运行 `solver`；和实验一一样，每个方法前后都会执行 `Flow.Analysis.preprocess` 和 `Flow.Analysis.postprocess`。
2. 使用 `QuadIterator` 或 `BasicBlock` 对代码内容进行修改，包括 `QuadIterator.add()`、`QuadIterator.remove()`、`BasicBlock.removeQuad()`、`BasicBlock.addQuad()` 等。如果你需要修改流图结构，请查看 `ControlFlowGraph` 的相关方法。