Gentest User Guide

Set up

For java project, download jar file here.

For maven project, configure your pom.xml as follow:

```
<dependency>
	<groupId>tzuyu-project</groupId>
	<artifactId>gentest</artifactId>
	<version>0.0.1-SNAPSHOT</version>
</dependency>
```

Add tzuyu repository:

```
<repository>
	<id>tzuyu</id>
	<url>http://202.94.70.100:8081/nexus/content/repositories/snapshots</url>
</repository>
```

Gentest:

In Gentest, we aim to generate automatically valid statements in order to execute certain methods. To do that, values of method arguments, as well as instance of methods' receiver will be initialized based on their types.

Gentest handles 2 scenarios of test case generation:

- Generate test cases for a method or a fixed, ordered sequence of methods (methods can belong to different classes).
- Generate testcases for random selected sequences of methods (methods can belong to different classes).

Generate testcases for fixed sequence of methods:

Using `FixTraceTester`:

```
int numberOfTcs = ...;
FixTraceTester tester = new FixTraceTester(numberOfTcs);
Class<?> targetClass = ...;
Method method = targetClass.getMethod("method_name", …);
MethodCall methodcall = MethodCall.of(method, targetClass);
Pair<List<Sequence>, List<Sequence>> testcases = tester.test(Arrays
                  .asList(methodcall));
```

Or you can use the `FixTraceGentestBuilder` like this:

```
int numberOfTcs = 100;
Class<SampleProgram> targetClazz = SampleProgram.class;
```

```
FixTraceGentestBuilder builder = new FixTraceGentestBuilder(numberOfTcs);
builder.forClass(targetClazz)
            .method("method_name/signature")
            .method("method_name/signature");
Pair<List<Sequence>, List<Sequence>> testcases = builder.generate();
```

Generate testcases for random sequence of methods:

Using `RandomTester/ RandomTraceGentestBuilder`:

```
int numberOfTcs = …;
int queryMaxLength = …;
int testPerQuery = …;
RandomTester tester = new RandomTester(queryMaxLength, testPerQuery,
                                                    numberOfTcs);
Class<?> targetClass = …;
Method method1 = …;
MethodCall methodCall1 = MethodCall.of(method1, targetClass);
Method method2 = …;
MethodCall methodCall2 = MethodCall.of(method2 , targetClass);
List<MethodCall> methodCalls = new ArrayList<MethodCall>();
methodCalls.add(methodCall1);
methodCalls.add(methodCall2);
Pair<List<Sequence>, List<Sequence>> testcases = tester.test(methodCalls);
```

`RandomTraceGentestBuilder`

```
int numberOfTcs = 10;
Class<SampleProgram> targetClazz = SampleProgram.class;
RandomTraceGentestBuilder builder = new RandomTraceGentestBuilder(numberOfTcs);
int queryMaxLength = 3;
int testPerQuery = 2;
builder.forClass(targetClazz)
            .testPerQuery(testPerQuery)
            .queryMaxLength(queryMaxLength);
Pair<List<Sequence>, List<Sequence>> testcases = builder.generate();
```

Print the generated testcases:

```
//path to target source folder for test
String sourceFolder = getSrcFolder();
// in case you want to write test files into a same package, just set
// failPackage to null.
String passPackage = "pass.package.name.here";
String failPackage = "fail.package.name.here";
String methodPrefix = "testMethodName"; //prefix of generated test method name
String testclassPrefix = "TestClass"; //prefix of generated test class name
TestsPrinter printer = new TestsPrinter(passPackage, failPackage,
            methodPrefix, testclassPrefix, sourceFolder);
printer.printTests(testcases);
```

In `TestPrinter`, we convert testcases presented as our `Sequence` objects to `javaParser`'s compilation units by `JWriter`, then print those compilation units to file using `FileCompilationUnitPrinter` by

default. You can use your customized `CompilationUnitPrinter` instead of the default one, by using another constructor of `TestsPrinter`, as in the example below:

```
public void printTc(String passPackage, String failPackage,
      String methodPrefix, String testclassPrefix,
      Pair<List<Sequence>, List<Sequence>> testcases) {
      ICompilationUnitPrinter cuPrinter = new CustomizedCuPrinter();
      TestsPrinter printer = new TestsPrinter(passPackage,
            failPackage, methodPrefix, testclassPrefix, cuPrinter);
      printer.printTests(testcases);
}

private static class CustomizedCuPrinter implements ICompilationUnitPrinter {

      public void print(List<CompilationUnit> compilationUnits) {
      // do something
      }

}
```

Or even if you only want to convert `Sequences` to `CompilationUnits`, call `JWriter`.