EN.601.422 / EN.601.622

Software Testing & Debugging

Plan for Today

- Unit Testing Randomness
- Unit Testing methods that do I/O
- RESTful API Testing

Test Randomness

- We need to distinguish:
  - How to test a (pseudo)random generator?
  - How to test a program that makes use of a (pseudo)random generator

How to test a (pseudo)random generator?

- How to test a (pseudo)random generator?
  - In some applications, such as cryptography, it's not enough that the random number generator works to produce an apparently random sequence, it has to be really random
  - A truly random sequence 1) must be *unpredictable* and 2) *cannot be reliably reproduced*.
  - Need statistical tests e.g., chi-square test to verify this
    - Outside the scope of our discussion

How to test a program that makes use of a pseudo-random generator

- 1. Control for the “seed” value in pseudo-random generators
  - Repeatability
- 2. Probe the range
  - Call the pseudo-random generator (a number of times) and verify the actual outputs are within the expected range
- 3. Probe the distribution
  - e.g., uniform distribution: all digits should occur with (roughly) equal frequency over a sufficiently long period of time
- 4. Use a deterministic pseudo-random generator

Set the seed value

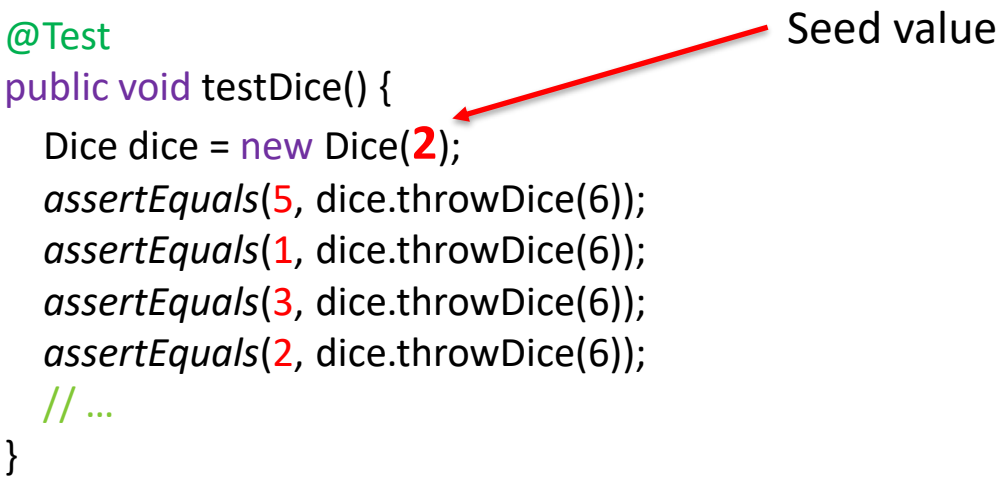
- The algorithm of pseudo-random generator use a recursive routine starting from a base value that is determined by a value named the **"seed"**.
- Seed value “locks” the pseudo-random generator so that it produces the same exact sequence of values on every separate run.
- Thus, setting the seed value allows for replicability of runs

Example

```
public class Dice {
   private final int DEFAULT_SEED = 0;
   Random rnd;
   public Dice(Random rnd) {
      this.rnd = rnd;
   }
   public Dice() {
      this.rnd = new Random(DEFAULT_SEED);
   }
   public Dice(int seed) {
      this.rnd = new Random(seed);
   }
   public int throwDice(int bound) {
      return rnd.nextInt(bound) + 1;
   }
}
```

Set the seed value

- Using a fixed seed value, a fixed sequence is generated every single time



Probe the range

```
@Test
public void testDiceRange() {
    Dice dice = new Dice(2);
    for (int i = 0; i < 100; ++i) {
        int diceVal = dice.throwDice(6);
        assertTrue(diceVal <= 6 && diceVal >= 1);
    }
}
```

Using a deterministic (Pseudo)Random

```
public class DeterministicRandom extends Random {
  int sequentialNum = 0;

  public DeterministicRandom() {
    super();
  }

  public int nextInt() {
    return sequentialNum++;
  }
}

// Sample usage:
Dice dice = new Dice(new DeterministicRandom());
```

Using a deterministic (Pseudo)Random

```
@Test
public void testDiceDeterministicRandom() {
    Dice dice = new Dice(new DeterministicRandom());
    assertTrue(dice.throwDice(6) == 1);
    assertTrue(dice.throwDice(6) == 2);
    assertTrue(dice.throwDice(6) == 3);
    // so on so forth . . .
}
```

Unit Test Methods that do I/O

```
// defined in a class named MyIOUnit
public List<String> read(InputStream input) throws IOException {
   List<String> tokens = new ArrayList<>();
   StringBuilder builder = new StringBuilder();

   int data = input.read(); // read one byte of data
   while (data != -1) { // as long as there are bytes to read
      if (((char)data) != ',') { // has not reached end of token
         builder.append((char) data);
      } else {
         tokens.add(builder.toString().trim()); // add token to list
         builder.delete(0, builder.length());
      }
      data = input.read();  // read next byte of data
   }
   return tokens;
}
```

Unit Test Methods that do I/O

```
@Test
public void testRead() throws IOException {
   MyIOUnit unit = new MyIOUnit();
   // set up the input
   byte[] data = "123,456,789,123,456,789".getBytes();
   InputStream input = new ByteArrayInputStream(data);

   // call the method that does inputting
   List<String> tokens = unit.read(input);

   // verify results
   assertEquals("123", tokens.get(0));
   assertEquals("456", tokens.get(1));
   assertEquals("789", tokens.get(2));
   assertEquals("123", tokens.get(3));
   assertEquals("456", tokens.get(4));
   assertEquals("789", tokens.get(5));
}
```

Unit Test Methods that do I/O

```
// defined in a class named MyIOUnit
public void write(OutputStream output, List<String> tokens) throws IOException {
   for(int i = 0; i < tokens.size(); i++){
      if(i > 0) {
         output.write(',');
      }
      output.write(tokens.get(i).getBytes());
   }
}
```

Unit Test Methods that do I/O

```
@Test
public void testWrite() throws IOException {
   MyIOUnit unit = new MyIOUnit();
   ByteArrayOutputStream output = new ByteArrayOutputStream();

   List<String> tokens = new ArrayList<>();
   tokens.add("one");
   tokens.add("two");
   tokens.add("three");

   unit.write(output, tokens);

   String string = new String(output.toByteArray());
   assertEquals("one,two,three", string);
}
```

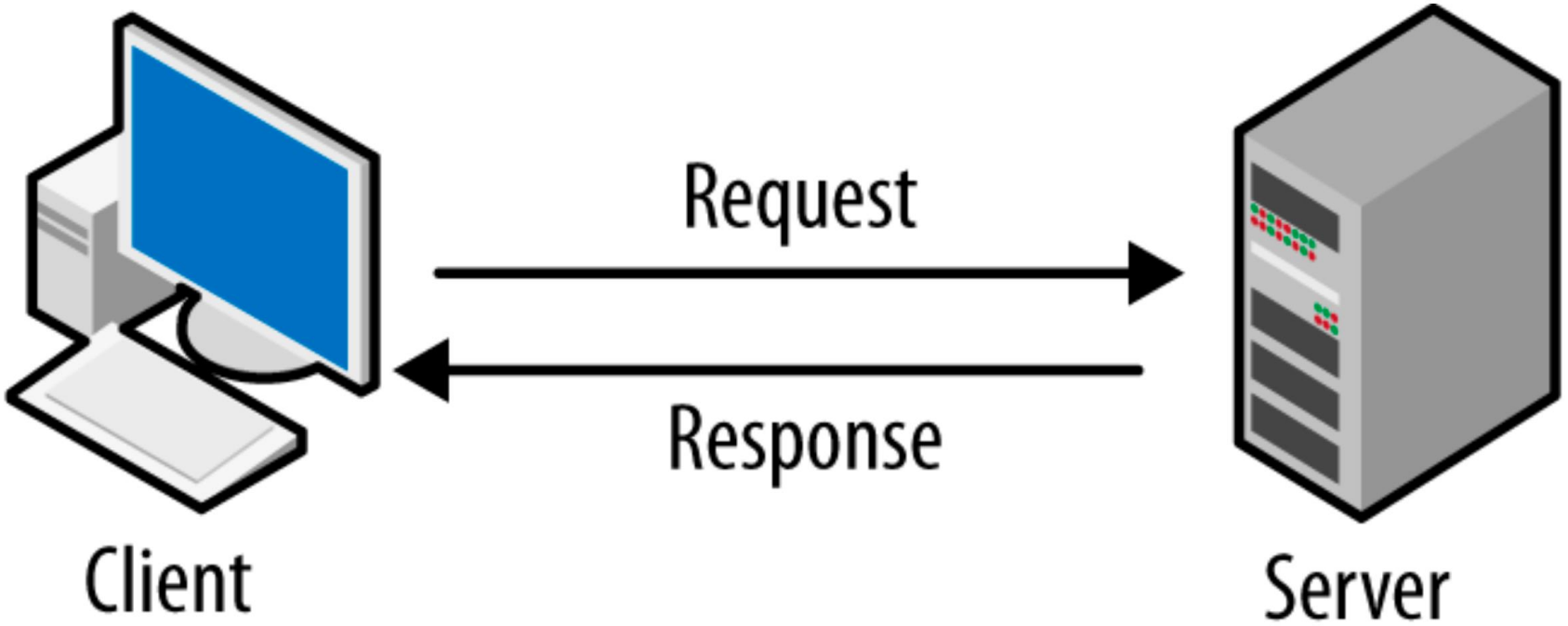
Unit Testing methods that do Standard I/O

```
// defined in a class named MyIOUnit
public void standardIO()  {
    Scanner scnr = new Scanner(System.in);
    String st = scnr.next();
    if (st.length() > 2) {
        System.out.print("long string");
    } else {
        System.out.print("short string");
    }
}
```

Unit Testing methods that do Standard I/O

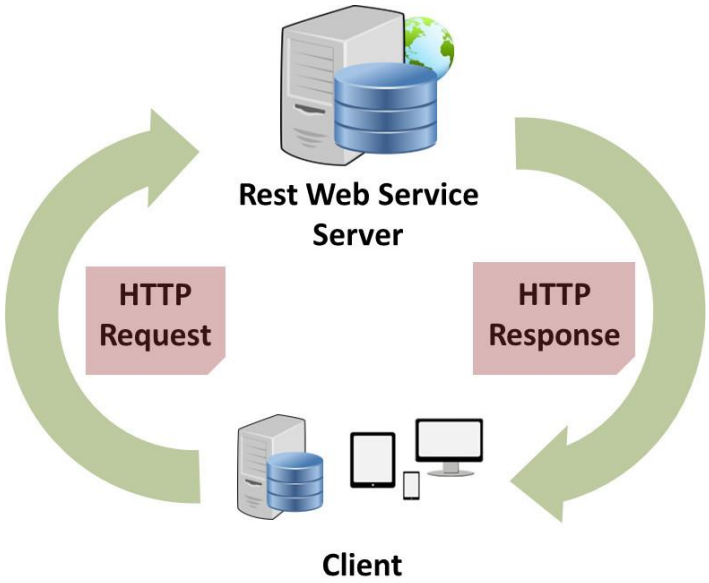
```
@Test
public void testStandardIO() {
   MyIOUnit unit = new MyIOUnit();
   // set up testable input and output channels
   byte[] data = "software testing".getBytes(StandardCharsets.UTF_8);
   BufferedInputStream in = new BufferedInputStream(new ByteArrayInputStream(data));
   ByteArrayOutputStream out = new ByteArrayOutputStream();
   PrintStream output = new PrintStream(out);
   // redirect the standard channels
   System.setIn(in);    System.setOut(output);
   // call to method that does standard I/O
   unit.standardIO();
   String string = out.toString();
   assertEquals("long string", string);
   // redirect the standard channels back!
   System.setIn(System.in);    System.setOut(System.out);
}
```

REST API Testing



REST API Testing

- Selenium used for client-side (front-end) testing
- The Server-side (backend) is comprised of API endpoints



Rest API Testing

- Test API end-points to verify how they handle and respond to http requests:
  - GET: fetch/read a resource
  - POST: save/create a resource
  - DELETE: remove a resource
  - PUT: update a resource

REST API Testing Tools

- Unit Testing:
  - Java: OKHttp, UniRest, Rest Assured
  - Python: pyresttest
- Dedicated REST API Testing/Development
  - Postman, Hoppscotch, Eggplant