Lecture 17: Unit Test Generation

CS 5150, Spring 2026

Lecture Goals

- Use automated techniques for unit-test generation
- Using coverage and mutation testing techniques

Unit Test Generation

Leveraging the Specifications

Automated Test Generation: Key Idea

Leverage the specifications to guide test generation:

- Types
- Invariants
- Pre- and Post- Conditions

Example: Leveraging Types

```
void remove(BinaryTree bt, Node n) {
  ...  // remove node n from binary tree bt
}
```

```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```

- Helps to avoid testing the remove method on arbitrary byte arrays

Example: Leveraging Invariants

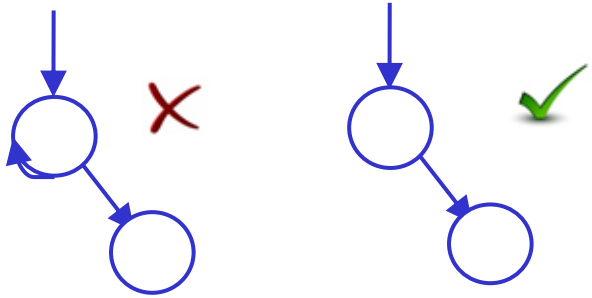
- Root may be null
- If root is not null:
  - No cycles
  - Each node (except root) has one parent
  - Root has no parent

```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```

Example: Leveraging Invariants

```
public boolean repOK(BinaryTree bt) {
   if (bt.root == null) return true;
   Set visited = new HashSet();
   List workList = new LinkedList();
   visited.add(bt.root);
   workList.add(bt.root);
   while (!workList.isEmpty()) {
      Node current = workList.removeFirst();
      if (current.left != null) {
         if (!visited.add(current.left)) return false;
         workList.add(current.left);
      }
      ... // similarly for current.right
   }
   return true;
}
```

```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```



Example: Leveraging Invariants

```
@invariant repOk(bt)

void remove(BinaryTree bt, Node n) {
  ...  // remove node n from binary tree bt
}
```

```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```

- Helps to avoid testing the remove method on non-tree structures
- Also serves as a contract to check at the end of remove method

Example: Leveraging Pre- and Post-Conditions

```
@invariant repOk(bt)
@requires contains(bt, n) // pre condition
@ensures !contains(bt, n) // post condition

void remove(BinaryTree bt, Node n) {
  ...  // remove node n from binary tree bt
}
```

```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```

- Helps to test even richer states on entry and exit of remove method

Testing Classes and Libraries

Key Ideas of Randoop

Randoop: Feedback-Directed Random Testing

How do we generate a test like this?

```
public static void test() {
   LinkedList l1 = new LinkedList();
   Object o1 = new Object();
   l1.addFirst(o1);
   TreeSet t1 = new TreeSet(l1);
   Set s1 = Collections.unmodifiableSet(t1);

   // This assertion fails
   assert(s1.equals(s1));
}
```

public TreeSet(Collection c): Constructs a new, empty tree set, sorted according to the specified comparator. **All elements inserted into the set must be *mutually comparable*** by the specified comparator: comparator.compare(e1, e2) must not throw a ClassCastException for any elements e1 and e2 in the set**. If the user attempts to add an element to the set that violates this constraint, the add call will throw a ClassCastException.** https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html

Overview

Problem with uniform random testing: Creates too many illegal or redundant tests

Idea: Randomly create new test guided by feedback from previously created tests

test == method sequence

Recipe:

- Build new sequences incrementally, extending past sequences
- As soon as a sequence is created, execute it
- Use execution results to guide test generation towards sequences that create new object states

Randoop: Input and Output

Input:

- classes under test
- time limit
- set of contracts

e.g. “o.hashCode() throws no exception”

e.g. “o.equals(o) == true”

Output:

- contract-violating test cases

```
LinkedList l1 = new LinkedList();
Object o1 = new Object();
l1.addFirst(o1);
TreeSet t1 = new TreeSet(l1);
Set s1 = Collections.unmodifiableSet(t1);

assert(s1.equals(s1));
```

No contract violated up to here

fails when executed

SEGMENT

The Randoop Algorithm

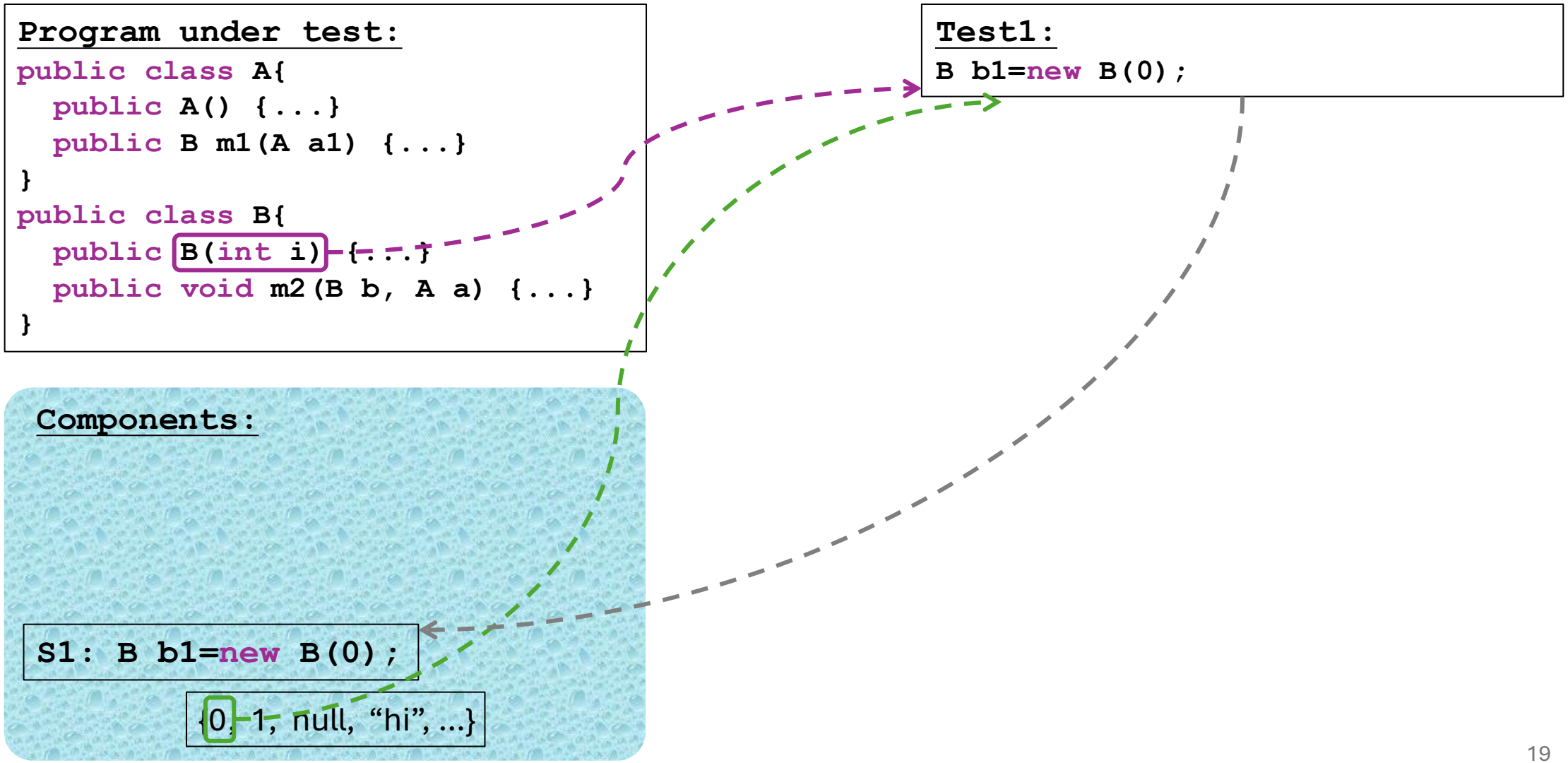
Randoop Algorithm

components = { int i = 0; boolean b = false; . . . } // seed components

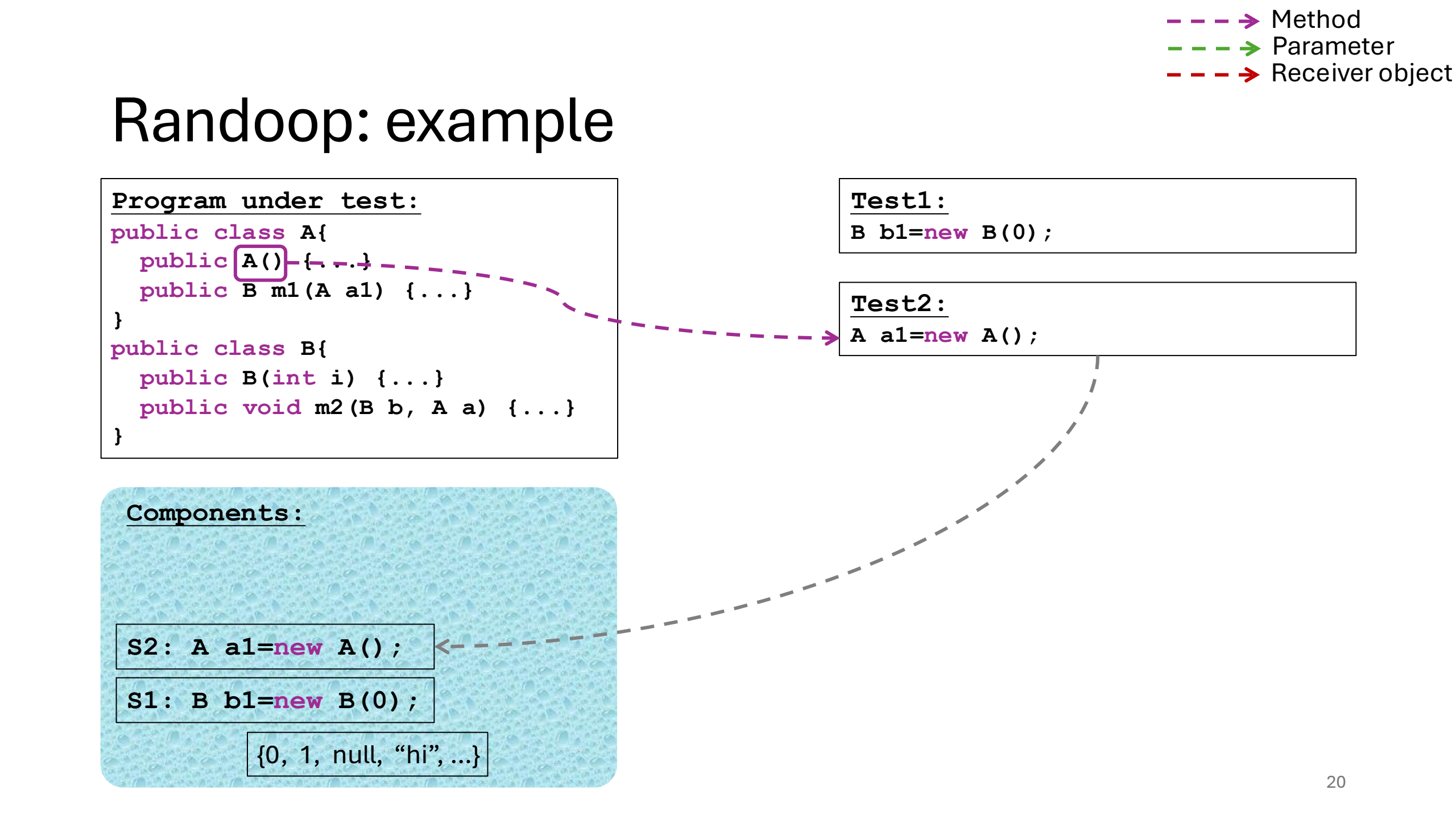
Repeat until time limit expires:

- Create a new sequence
  - Randomly pick a method call T_{ret} `m(`T_1`,...,`T_n`)`
  - For each argument of type T_i, randomly pick sequence S_i from components that constructs an object v_i of that type
  - Create S_{new} = S_1`; ...;` S_n`;` T_{ret} v_{new} `= m(`v_1`,...,`v_n`);`
- Classify new sequence S_{new}: discard / output as test / add to components

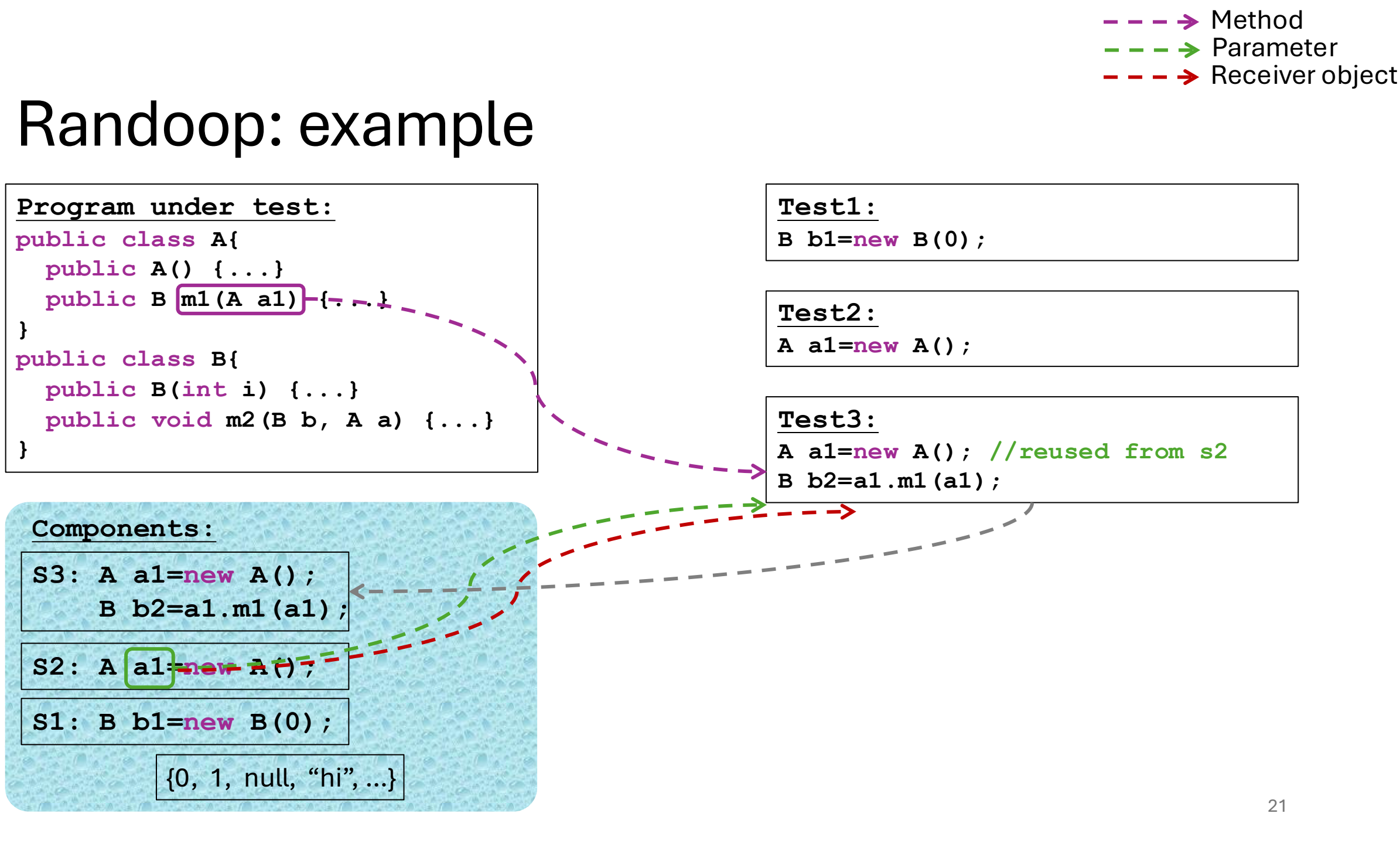
Randoop: example



Randoop: example



Randoop: example



Method
Parameter
Receiver object

Randoop: example

Program under test:

```
public class A{
  public A() {...}
  public B m1(A a1) {...}
}
public class B{
  public B(int i) {...}
  public void m2(B b, A a) {...}
}
```

Components:

```
S3: A a1=new A();
    B b2=a1.m1(a1);
```

```
S4: …
```

```
S2: A a1=new A();
```

```
S1: B b1=new B(0);
```

{0, 1, null, “hi”, …}

Test1:

```
B b1=new B(0);
```

Test2:

```
A a1=new A();
```

Test3:

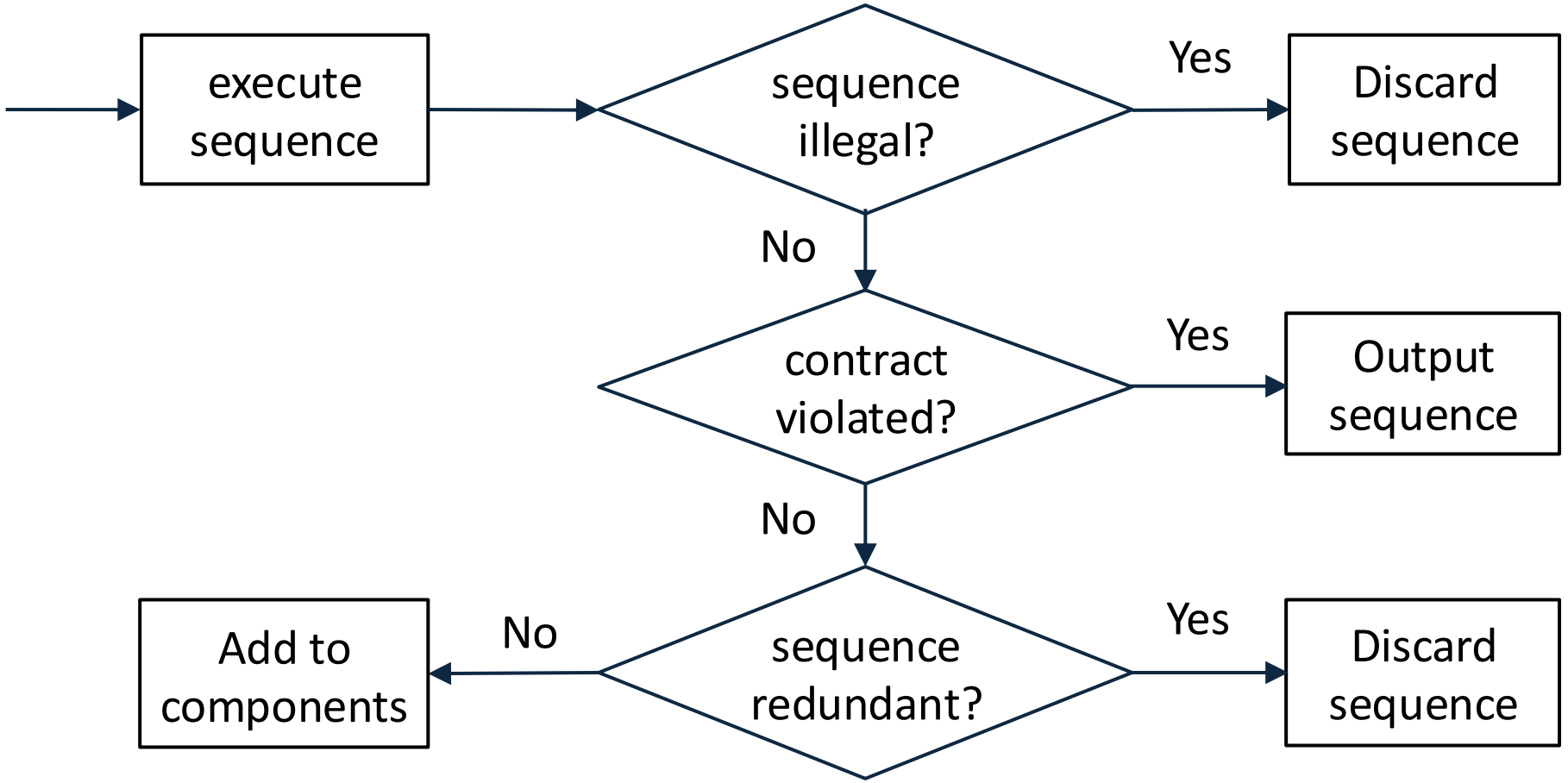
```
A a1=new A();
B b2=a1.m1(a1);
```

Test4:

```
B b1=new B(0); //reused from s1
A a1=new A();
B b2=a1.m1(a1); //reused from s3
b1.m2(b2, a1);
```

…

Classifying a Sequence



Illegal Sequences

- Sequences that “crash” before contract is checked
  - E.g., throw an exception

```
int i = -1;
Date d = new Date(2006, 2, 14);
d.setMonth(i);    // pre: argument >= 0
assert(d.equals(d));
```

Redundant Sequences

- Maintain set of all objects created in execution of each sequence
- New sequence is redundant if each object created during its execution belongs to above set (using equals to compare)
- Could also use more sophisticated state equivalence methods

```
Set s = new HashSet();
s.add(“hi”);


assertTrue(s.equals(s));
```

```
Set s = new HashSet();
s.add(“hi”);
s.isEmpty();

assertTrue(s.equals(s));
```

SEGMENT

Randoop in Practice

Code coverage by Randoop

Data structure programs	Time (s)	Branch cov.
Bounded stack (30 LOC)	1	100%
Unbounded stack (59 LOC)	1	100%
BS Tree (91 LOC)	1	96%
Binomial heap (309 LOC)	1	84%
Linked list (253 LOC)	1	100%
Tree map (370 LOC)	1	81%
Heap array (71 LOC)	1	100%

Bug detection by Randoop: subjects

Subjects	LOC	Classes
JDK (2 libraries) (java.util, javax.xml)	53K	272
Apache commons (6 libraries) (logging, primitives, chain, jelly, math, collections)	114K	974
.Net libraries (6 libraries)	615K	3455

Bug detection by Randoop: subjects

Subjects	Failed tests	Unique failed tests	Error-revealing tests	Distinct errors
JDK	613	32	29	8
Apache commons	3,044	187	29	6
.Net framework	543	205	196	196
Total	4,200	424	254	210

Some Bugs Found by Randoop

- JDK containers have 4 methods that violate `o.equals(o)` contract
- Javax.xml creates objects that cause `hashCode` and `toString` to crash, even though objects are well-formed XML constructs
- Apache libraries have constructors that leave fields unset, leading to NPE on calls of `equals`, `hashCode`, and `toString`
- .Net framework has at least 175 methods that throw an exception forbidden by the library specification (NPE, out-of-bounds, or illegal state exception)
- .Net framework has 8 methods that violate `o.equals(o)` contract

QUIZ: Randoop Test Generation (Part 1)

Write the smallest sequence that Randoop can possibly generate to create a valid BinaryTree.

Once generated, how does Randoop classify it?

- ◯ Discards it as illegal
- ◯ Outputs it as a bug
- ◯ Adds to components for future extension

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
     assert(repOk(this));
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```

QUIZ: Randoop Test Generation (Part 1)

Write the smallest sequence that Randoop can possibly generate to create a valid BinaryTree.

```
BinaryTree bt = new BinaryTree(null);
```

Once generated, how does Randoop classify it?

- ◯ Discards it as illegal
- ◯ Outputs it as a bug
- ◯ Adds to components for future extension

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
     assert(repOk(this));
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```

QUIZ: Randoop Test Generation (Part 2)

Write the smallest sequence that Randoop can possibly generate that violates the assertion in removeRoot().

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
     assert(repOk(this));
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```

Once generated, how does Randoop classify it?

- ◯ Discards it as illegal
- ◯ Outputs it as a bug
- ◯ Adds to components for future extension

QUIZ: Randoop Test Generation (Part 2)

Write the smallest sequence that Randoop can possibly generate that violates the assertion in removeRoot().

```
BinaryTree bt = new BinaryTree(null);
bt.removeRoot();
```

Once generated, how does Randoop classify it?

- ◯ Discards it as illegal
- ◯ Outputs it as a bug
- ◯ Adds to components for future extension

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
     assert(repOk(this));
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```

QUIZ: Randoop Test Generation (Part 3)

Write the smallest sequence that Randoop can possibly generate that violates the assertion in BinaryTree's constructor.

Can Randoop create a BinaryTree object with cycles using the given API?

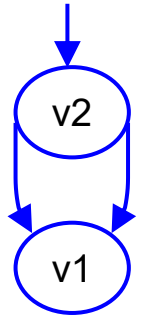
○ Yes ○ No

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
     assert(repOk(this));
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```

QUIZ: Randoop Test Generation (Part 3)

Write the smallest sequence that Randoop can possibly generate that violates the assertion in BinaryTree's constructor.




```
Node v1 = new Node(null, null);
Node v2 = new Node(v1, v1);
BinaryTree bt = new BinaryTree(v2);
```

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
     assert(repOk(this));
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

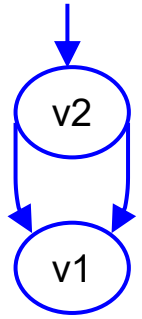
```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```

Can Randoop create a BinaryTree object with cycles using the given API?

○ Yes ○ No

QUIZ: Randoop Test Generation (Part 3)

Write the smallest sequence that Randoop can possibly generate that violates the assertion in BinaryTree's constructor.




```
Node v1 = new Node(null, null);
Node v2 = new Node(v1, v1);
BinaryTree bt = new BinaryTree(v2);
```

Can Randoop create a BinaryTree object with cycles using the given API?

○ Yes ● No

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
     assert(repOk(this));
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```