Chapter 7

Implementing a Listener

7.2 Printing the Search Events

We start with an example listener that simply prints the sequence of events that have been triggered during JPF's search. For each event, it prints the sort of event and the ID of the state of the search in which the event occurs. We name our class `SearchEvents`. It implements the interface `SearchListener`. Therefore, we have the following class header.

```
public class SearchEvents implements SearchListener
```

We implement each method of the `SearchListener` interface. For example, the `stateAdvanced` method is implemented as follows.

```
public void stateAdvanced(Search search) {
  System.out.println("advanced to state " + search.getStateId());
}
```

Recall that the `Search` object can be used to get information about the search. In particular, its `getStateId` returns the ID of the current state of the search.

For example, consider the following application (the comments in the code will become clear later).

```
import java.util.Random;

public class StateSpace {
  // state -1
  public static void main(String[] args) {
    // state 0
    Random random = new Random();
    if (!random.nextBoolean()) {
      // state 1
    } else {
      // state 2
      if (!random.nextBoolean()) {
        // state 1
      } else {
        // state 1
      }
    }
  }
}
```

If we run JPF on the above application, then the `SearchEvents` listener produces the following output.[1]

```
1  search started in state -1
2  advanced to state 0
3  advanced to state 1
4  state 1 processed
5  backtracked to state 0
6  advanced to state 2
7  advanced to state 1
8  backtracked to state 2
9  advanced to state 1
10 backtracked to state 2
11 state 2 processed
12 backtracked to state 0
13 state 0 processed
14 backtracked to state -1
15 state -1 processed
16 search finished in state -1
```

The state-transition diagram is depicted in Figure 7.1. The search starts in state -1 (line 1). Subsequently, it advances to state 0 (line 2) and then to state 1 (line 3). State 1 is a final state. Since state 1 has no outgoing transitions (that have not been traversed yet), the search signals that the state has been processed (line 4). Next, it backtracks to state 0 (line 5). From state 0, the search advances along the still unexplored transition to state 2 (line 6). From state 2 it advances to state 1 (line 7). This state has already been fully explored so the search backtracks to state 2 (line 8). Since state 2 has another, still unexplored, transition to state 1, the search advances along this transition to state 1 (line 9). Next, the search backtracks to state 2 (line 10), after which it signals that this state has been processed (line 11), and similarly for state 0 (line 12–13) and state -1 (line 14–15). At that point, the search terminates (line 16).

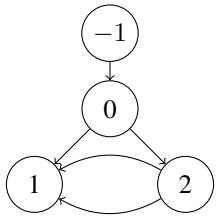

Figure 7.1: state-transition diagram of `StateSpace`.

State 0 is often considered the initial state. The bytecode instructions associated with the transition from state -1 to state 0 correspond to initializing the virtual machine and invoking the `main` method.

If we use breadth-first search for the above application, then the `SearchEvents` listener produces the following output.[2]

```
1 search started in state -1
2 state -1 stored
3 state -1 restored
4 advanced to state 0
5 state 0 stored
6 backtracked to state -1
```

[1] The `DFSearch` discussed in Chapter **??** does not produce all of the events (line 4 is missing), but the `DFSearch` that we will develop in Chapter **??** does.

[2] The `BFSHeuristic` discussed in Chapter **??** does not produce all of the events in this order (line 1 and 2 are reversed, and line 3 and 10 are missing), but the `BFSearch` that we will develop in Chapter **??** does.

```
state -1 processed
state 0 restored
advanced to state 1
state 1 processed
backtracked to state 0
advanced to state 2
state 2 stored
backtracked to state 0
state 0 processed
state 2 restored
advanced to state 1
backtracked to state 2
advanced to state 1
backtracked to state 2
state 2 processed
search finished in state 2
```

The main difference with depth-first search is that breadth-first search stores and restores states.