

The `run` method of the `Generator` class is modified as follows.

```
final int STOP = 5;
boolean done = false;
while (!done) {
    ...
    done = random.nextInt(STOP) == 0;
}
```

Whenever the **Generator** terminates, we want to print the sum of the integers it produced.

Question

Which changes have to be made to the **Listener** interface?

Whenever the **Generator** terminates, we want to print the sum of the integers it produced.

Question

Which changes have to be made to the **Listener** interface?

Answer

Add

```
void stop();
```

Whenever the **Generator** terminates, we want to print the sum of the integers it produced.

Question

Which changes have to be made to the **Generator** class?

Listener

Whenever the **Generator** terminates, we want to print the sum of the integers it produced.

Question

Which changes have to be made to the **Generator** class?

Answer

```
final int STOP = 5;
boolean done = false;
while (!done) {
    ...
    done = random.nextInt(STOP) == 0;
}
for (Listener listener : this.listeners) {
    listener.stop();
}
```

Whenever the **Generator** terminates, we want to print the sum of the integers it produced.

Question

Which changes have to be made to the **ListenerAdapter** class?

Whenever the **Generator** terminates, we want to print the sum of the integers it produced.

Question

Which changes have to be made to the **ListenerAdapter** class?

Answer

Add

```
public void stop() {  
    // default implementation  
}
```

SumPrinter

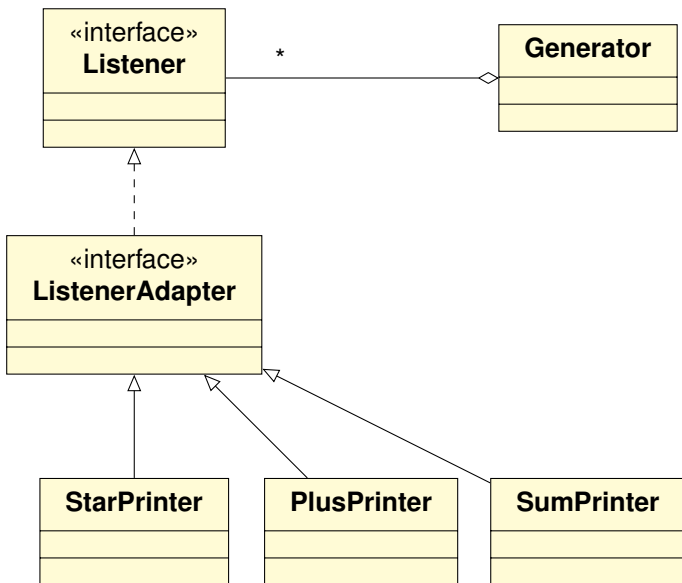
```
public class SumPrinter extends ListenerAdapter {
    private int sum;

    public SumPrinter() {
        this.sum = 0;
    }

    public void process(int value) {
        this.sum += value;
    }

    public void stop() {
        System.out.println("-----");
        System.out.println(this.sum);
        System.out.println("-----");
    }
}
```


Generator and Listener

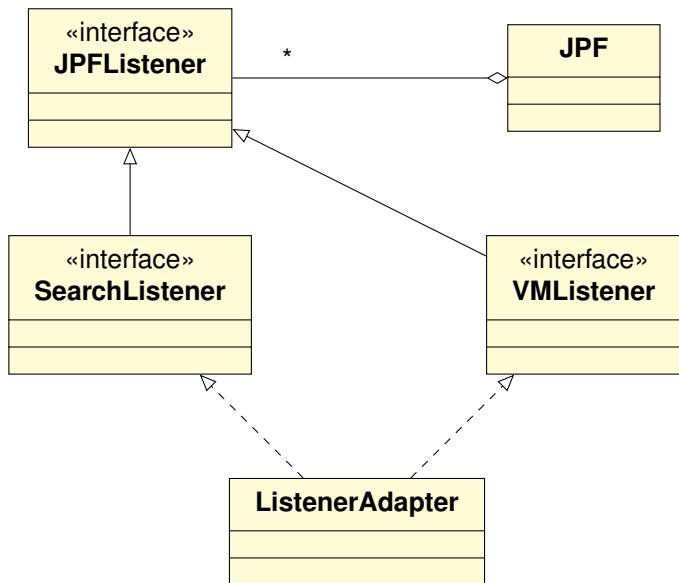


Listeners in JPF

EECS 4315

www.cse.yorku.ca/course/4315/

Generator and Listener



The interface **JPFListener** is empty.

The interface `JPFListener` is empty.

Question

Why introduce an empty interface?

The interface `JPFListener` is empty.

Question

Why introduce an empty interface?

Answer

For type checking.

```
public interface VMListener extends JPFListener {

    // VM has been initialized and, hence, classes
    // have been loaded
    void vmInitialized(VM vm);

    // A number of methods related to the execution
    // of instructions
    void executeInstruction (VM vm,
        ThreadInfo currentThread,
        Instruction instructionToExecute);
    ...

    // A number of methods related to threads
    void threadStarted(VM vm,
        ThreadInfo startedThread);
    ...
}
```

```
// Class has been loaded
void loadClass (VM vm, ClassFile cf);

// A number of methods related to objects
void objectCreated (VM vm, ThreadInfo currentThread,
    ElementInfo newObject);
...

// A number of methods related to garbage
// collection
void gcBegin (VM vm);
...
```



```
// A number of methods related to exceptions
void exceptionThrown (VM vm,
    ThreadInfo currentThread,
    ElementInfo thrownException);
...

// A number of methods related to choice
// generators
void choiceGeneratorRegistered (VM vm,
    ChoiceGenerator<?> nextCG,
    ThreadInfo currentThread,
    Instruction executedInstruction);
...
```

```
// A number of methods about methods
void methodEntered (VM vm,
    ThreadInfo currentThread,
    MethodInfo enteredMethod);
...
}
```

Write a listener that print a * whenever the garbage collector is invoked by JPF.

Write a listener that measures the amount of time (in milliseconds) JPF's garbage collector takes whenever it is invoked by JPF.

```
public interface SearchListener extends JPFLListener
{
    void stateAdvanced(Search search);
    void stateProcessed(Search search);
    void stateBacktracked(Search search);
    void statePurged(Search search);
    void stateStored(Search search);
    void stateRestored(Search search);
    void propertyViolated(Search search);
    void searchStarted(Search search);
    void searchConstraintHit(Search search);
    void searchFinished(Search search);
}
```

Implement a listener which prints the states and transitions visited by the search in the following simple format:

0 -> 1

1 -> 2

0 -> 3

3 -> 4

4 -> 2

Implement a listener which creates a dot file representing the the states and transitions visited by the search.

```
digraph statespace {  
0 -> 1  
1 -> 2  
0 -> 3  
3 -> 4  
4 -> 2  
}
```

Implement a listener which creates a dot file representing the the states and transitions visited by the search. Colour the initial state green and the final states red.

```
digraph statespace {  
0 [fillcolor=green]  
0 -> 1  
1 -> 2  
2 [fillcolor=red]  
0 -> 3  
3 -> 4  
4 -> 2  
}
```