

Listen
EECS 4315

`wiki.eecs.yorku.ca/course/4315/`



JPF uses (event) listeners.

```
target=Traversal  
classpath=.  
cg.enumerate_random=true  
listener=gov.nasa.jpf.listener.StateSpaceDot
```

The method `run` of the class `Generator` produces integer values. On average, it produces an integer value every two seconds (according to a Gaussian distribution with a mean of two seconds and a standard deviation of one second). It produces integers in the interval $[0, 9]$ uniformly at random.

```
public class Generator {
    public void run() {
        Random random = new Random();
        final int MEAN_DELAY = 2000;
        final int SD_DELAY = 1000;
        final int MAX_VALUE = 9;
        while (true) {
            int delay = MEAN_DELAY +
                (int) (SD_DELAY * random.nextGaussian());
            try {
                Thread.sleep(delay);
            } catch (InterruptedException e) {}
            int value = random.nextInt(MAX_VALUE + 1);
        }
    }
}
```

The `Main` app creates a `Generator` object and invokes its `run` method.

```
public class Main {  
    public static void main(String[] args) {  
        Generator generator = new Generator();  
        generator.run();  
    }  
}
```

Whenever the **Generator** produces an integer, we want to process it. For example, we can print *. We want to **decouple** the processing of the integers from the production of the integers so that we need not make any changes to the **Generator** class if we want to change the processing of the integers. Hence, we create a **StarPrinter** class with a method **process** to print *.


```
public class StarPrinter {  
    public void process() {  
        System.out.println("*");  
    }  
}
```

Event generator and listener

Whenever the **Generator** produces an integer, it should invoke the **process** method on a **StarPrinter** object.

```
public class Generator {
    public void run() {
        ...
        while (true) {
            ...
            int value = random.nextInt(...);
            ????.process();
        }
    }
}
```

Question

How do we store the reference ??? to a `StarPrinter` object in the `Generator` class?

Event generator and listener

Question

How do we store the reference ??? to a `StarPrinter` object in the `Generator` class?

Answer

As an attribute.

Event generator and listener

Question

How do we store the reference ??? to a `StarPrinter` object in the `Generator` class?

Answer

As an attribute.

```
public class Generator {  
    private ??? x;  
    public void run() {  
        ...  
        while (true) {  
            ...  
            this.x.process();  
        }  
    }  
}
```

Question

What is the type of the attribute `x`?

Event generator and listener

Question

What is the type of the attribute `x`?

Answer

`StarPrinter.`

Event generator and listener

Question

What is the type of the attribute `x`?

Answer

`StarPrinter`.

```
public class Generator {  
    private StarPrinter x;  
    public void run() {  
        ...  
        while (true) {  
            ...  
            this.x.process();  
        }  
    }  
}
```



```
public class PlusPrinter {  
    public void process() {  
        System.out.println("+");  
    }  
}
```

Event generator and listener

```
public class PlusPrinter {  
    public void process() {  
        System.out.println("+");  
    }  
}
```

Question

How can we modify the type of the attribute `x` and the classes `StarPrinter` and `PlusPrinter` so that the class `Generator` can use both?

Event generator and listener

```
public class PlusPrinter {  
    public void process() {  
        System.out.println("+");  
    }  
}
```

Question

How can we modify the type of the attribute `x` and the classes `StarPrinter` and `PlusPrinter` so that the class `Generator` can use both?

Answer

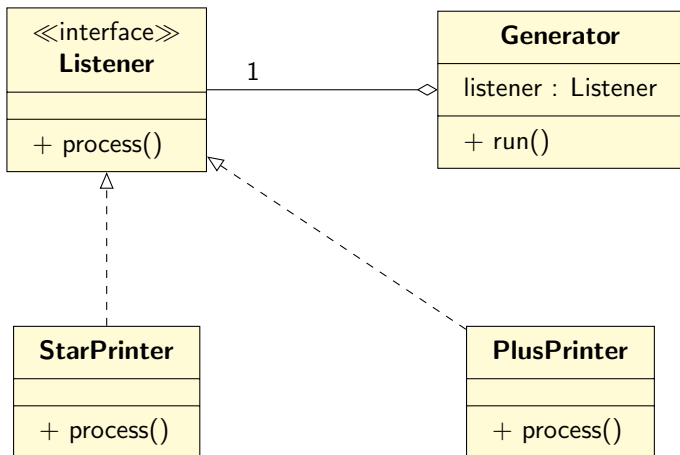
Introduce an interface `Listener`, change the type of the attribute `x` to `Listener`, and specify that the classes `StarPrinter` and `PlusPrinter` implement `Listener`.

```
public interface Listener {  
    void process();  
}
```

```
public class Generator {  
    private Listener listener;  
    public void run() {  
        ...  
        while (true) {  
            ...  
            this.listener.process();  
        }  
    }  
}
```

```
public class StarPrinter implements Listener {  
    public void process() {  
        System.out.println("*");  
    }  
}
```

Generator and listener



Question

How do we initialize the `listener` attribute of the `Generator` class?

Question

How do we initialize the `listener` attribute of the `Generator` class?

Answer

By means of a mutator `setListener`.

Question

How do we initialize the `listener` attribute of the `Generator` class?

Answer

By means of a mutator `setListener`.

```
public class Generator {  
    private Listener listener;  
    public void setListener(Listener listener) {  
        this.listener = listener;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Generator generator = new Generator();  
        Listener starPrinter = new StarPrinter();  
        generator.setListener(starPrinter);  
        generator.run();  
    }  
}
```

Question

Which changes do we have to make if we want to associate multiple listeners with the generator? For example, we would like a * and + to be printed whenever an integer is produced.

Multiple listeners

Question

Which changes do we have to make if we want to associate multiple listeners with the generator? For example, we would like a * and + to be printed whenever an integer is produced.

Answer

Instead of an attribute that represents a `Listener`, use an attribute that represents a collection of `Listeners`.

Question

Instead of

```
private Listener listener;
```

what do we use to represent a collection of **Listeners**?

Multiple listeners

Question

Instead of

```
private Listener listener;
```

what do we use to represent a collection of **Listeners**?

Answer

```
private List<Listener> listeners;
```

Question

Where and how do we initialize the attribute `listeners`?

Question

Where and how do we initialize the attribute `listeners`?

Answer

```
public Generator() {  
    this.listeners = new ArrayList<Listener>();  
}
```

Question

How do we add a listener to the `listeners`?

Multiple listeners

Question

How do we add a listener to the `listeners`?

Answer

```
public void addListener(Listener listener) {  
    this.listeners.add(listener);  
}
```

Question

How do we invoke the `process` method on the `listeners`?

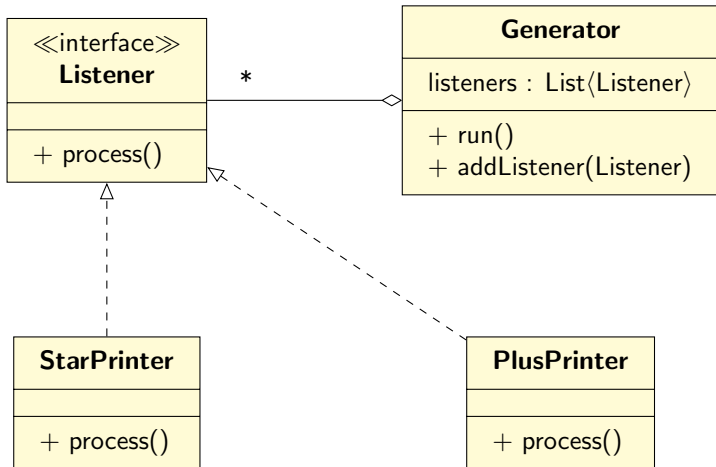
Question

How do we invoke the `process` method on the `listeners`?

Answer

```
for (Listener listener : this.listeners) {  
    listener.process();  
}
```

Generator and listeners



Whenever the **Generator** produces an integer, we want to print it.

Whenever the **Generator** produces an integer, we want to print it.

Question

How does the **Generator** pass the produced integer to the **Listener**?

Whenever the **Generator** produces an integer, we want to print it.

Question

How does the **Generator** pass the produced integer to the **Listener**?

Answer

Pass the produced integer as an argument.

```
public void process(int value) {  
    ...  
}
```

```
public interface Listener {  
    void process();  
    void process(int value);  
}
```

```
public class ValuePrinter implements Listener {  
    public void process() {  
        ???  
    }  
  
    public void process(int value) {  
        System.out.println(value);  
    }  
}
```

Question

Since the class `ValuePrinter` implements the interface `Listener`, it has to provide an implementation of `process()` and `process(int)`. How to implement `process()`?

```
public class ValuePrinter implements Listener {  
    public void process() {  
        ???  
    }  
  
    public void process(int value) {  
        System.out.println(value);  
    }  
}
```

Question

Since the class `ValuePrinter` implements the interface `Listener`, it has to provide an implementation of `process()` and `process(int)`. How to implement `process()`?

Answer

```
public void process() {}
```

```
public class StarPrinter implements Listener {  
    public void process() {  
        System.out.println("*");  
    }  
  
    public void process(int value) {  
        ???  
    }  
}
```

Question

Since the class `StarPrinter` implements the interface `Listener`, it has to provide an implementation of `process()` and `process(int)`. How to implement `process(int)`?

```
public class StarPrinter implements Listener {  
    public void process() {  
        System.out.println("*");  
    }  
  
    public void process(int value) {  
        ???  
    }  
}
```

Question

Since the class `StarPrinter` implements the interface `Listener`, it has to provide an implementation of `process()` and `process(int)`. How to implement `process(int)`?

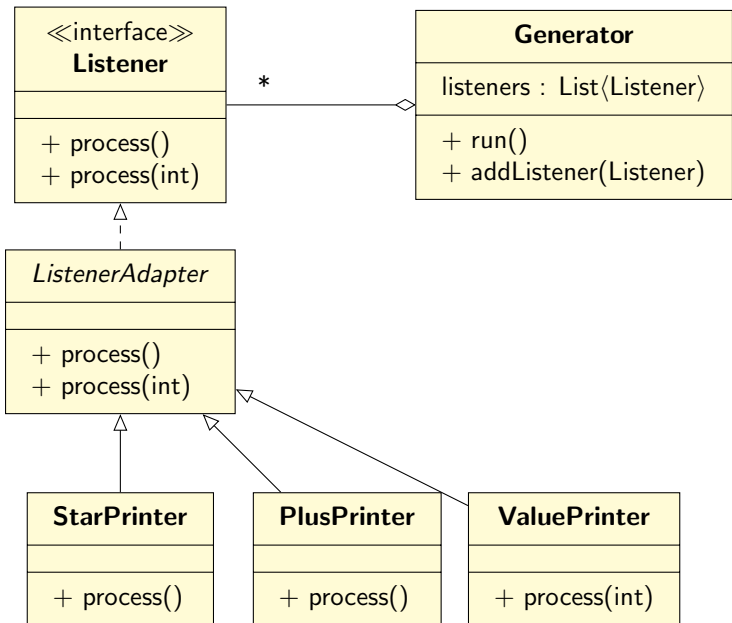
Answer

```
public void process(int value) {}
```

Rather than duplicating these default implementations in classes implementating the interface `Listener`, we introduce the class `ListenerAdapter` that contains a default implementation for each method.

```
public abstract class ListenerAdapter implements Listener {  
    public void process() {}  
    public void process(int value) {}  
}
```


Generator and listeners



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

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?

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  
}
```

Problem

Implement the `SumPrinter` class?

```
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 listeners

