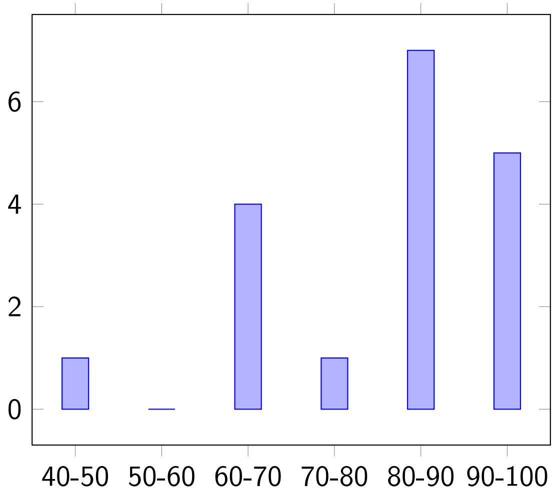
Quiz 1: grade distribution



Average: 83%

Grades description

A+ Exceptional. Thorough knowledge of concepts and/or techniques and exceptional skill or great originality in the use of those concepts/techniques in satisfying the requirements of an assignment or course.

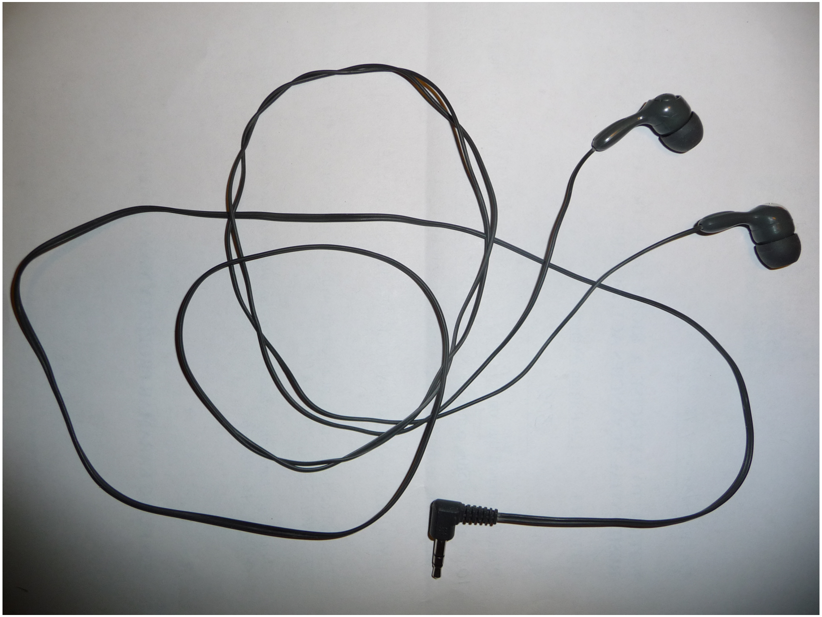
A Excellent. Thorough knowledge of concepts and/or techniques together with a high degree of skill and/or some elements of originality in satisfying the requirements of an assignment or course.

B+ Very Good. Thorough knowledge of concepts and/or techniques together with a fairly high degree of skill in the use of those concepts/techniques in satisfying the requirements of an assignment or course.

Listen

EECS 4315

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



Event listeners

JPF uses (event) listeners.

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

Event generator

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.

Event generator

```
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);
    }
  }
}
```

Main app

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

Main app

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

Event listener

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 $*$.

Event listener

```
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();
    }
  }
}
```

Event generator and listener

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();
    }
  }
}
```

Event generator and listener

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();
    }
  }
}
```

Event generator and listener

```
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`.

Listener

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

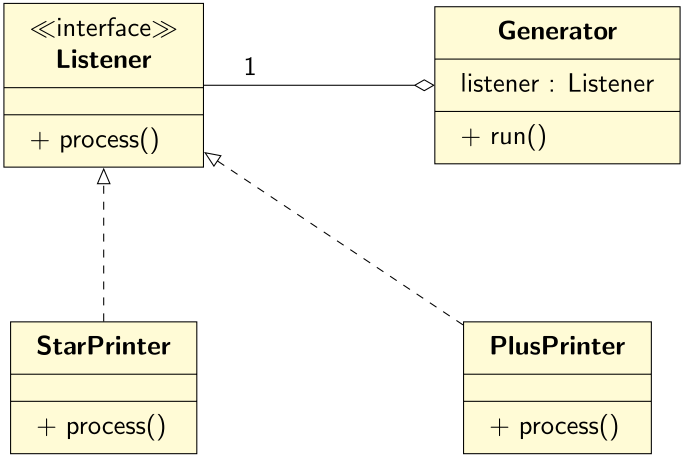
Generator

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

StarPrinter

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

Generator and listener



Generator

Question

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

Generator

Question

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

Answer

In the constructor.

Generator

Question

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

Answer

In the constructor.

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

Main app

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

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.

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 `Listener`s.

Multiple listeners

Question

Instead of

```
private Listener listener;
```

what do we use to represent a collection of `Listener`s?

Multiple listeners

Question

Instead of

```
private Listener listener;
```

what do we use to represent a collection of `Listener`s?

Answer

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

Multiple listeners

Question

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

Multiple listeners

Question

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

Answer

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

Multiple listeners

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);
}
```

Multiple listeners

Question

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

Multiple listeners

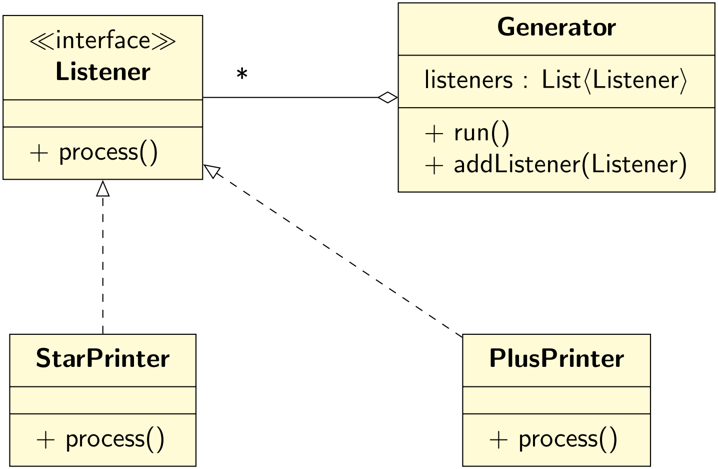
Question

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

Answer

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

Generator and listeners



Listener

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

Listener

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

Question

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

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) {
  ...
}
```

Listener

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

Listener

```
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()`?

Listener

```
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() {}
```

Listener

```
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)`?

Listener

```
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) {}`

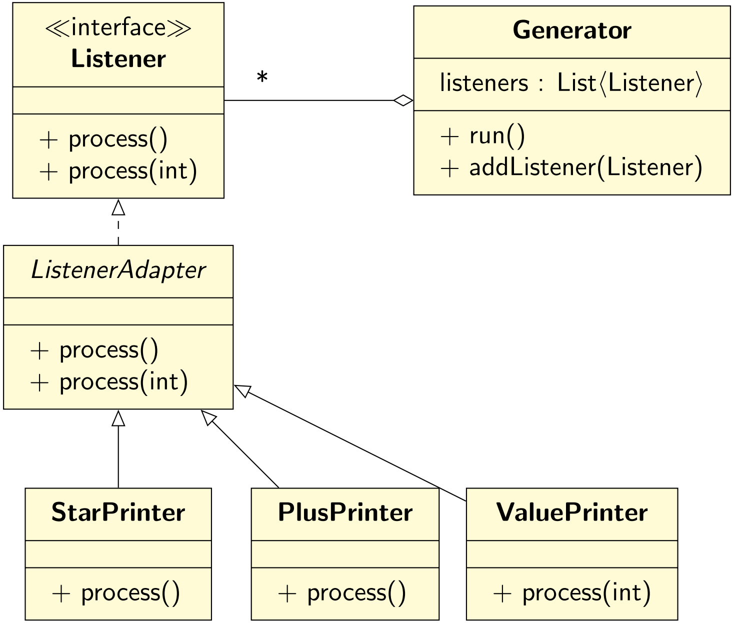
ListenerAdapter

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.

ListenerAdapter

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

Generator and listeners



Listener

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

Listener

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;
}
```

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 `Listener` interface?

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 `Listener` interface?

Answer

Add

```
void stop();
```

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?

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();
}
```

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 `ListenerAdapter` 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 `ListenerAdapter` class?

Answer

Add

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

Listener

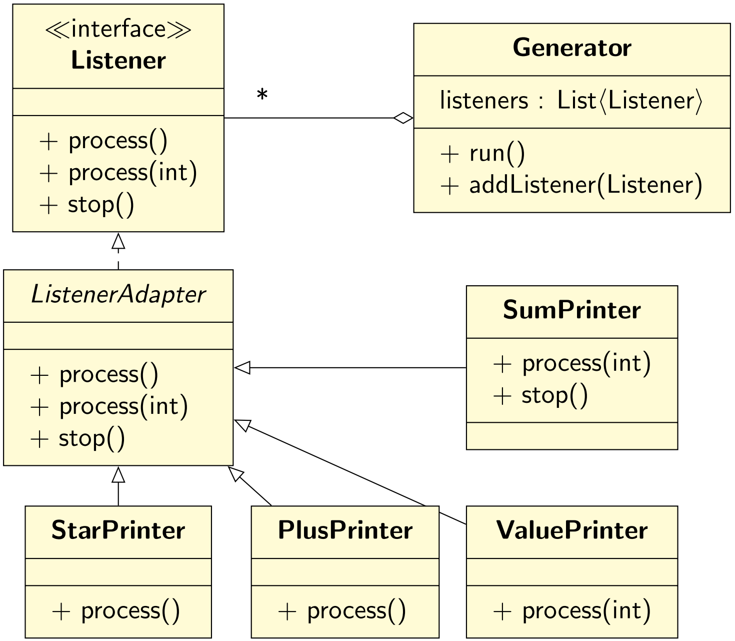
Problem

Implement the `SumPrinter` class?

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 listeners



≪interface≫
Listener
+ process()
+ process(int)
+ stop()
*
Generator
listeners : List⟨Listener⟩
+ run()
+ addListener(Listener)
ListenerAdapter
+ process()
+ process(int)
+ stop()
SumPrinter
+ process(int)
+ stop()
StarPrinter
+ process()
PlusPrinter
+ process()
ValuePrinter
+ process(int)