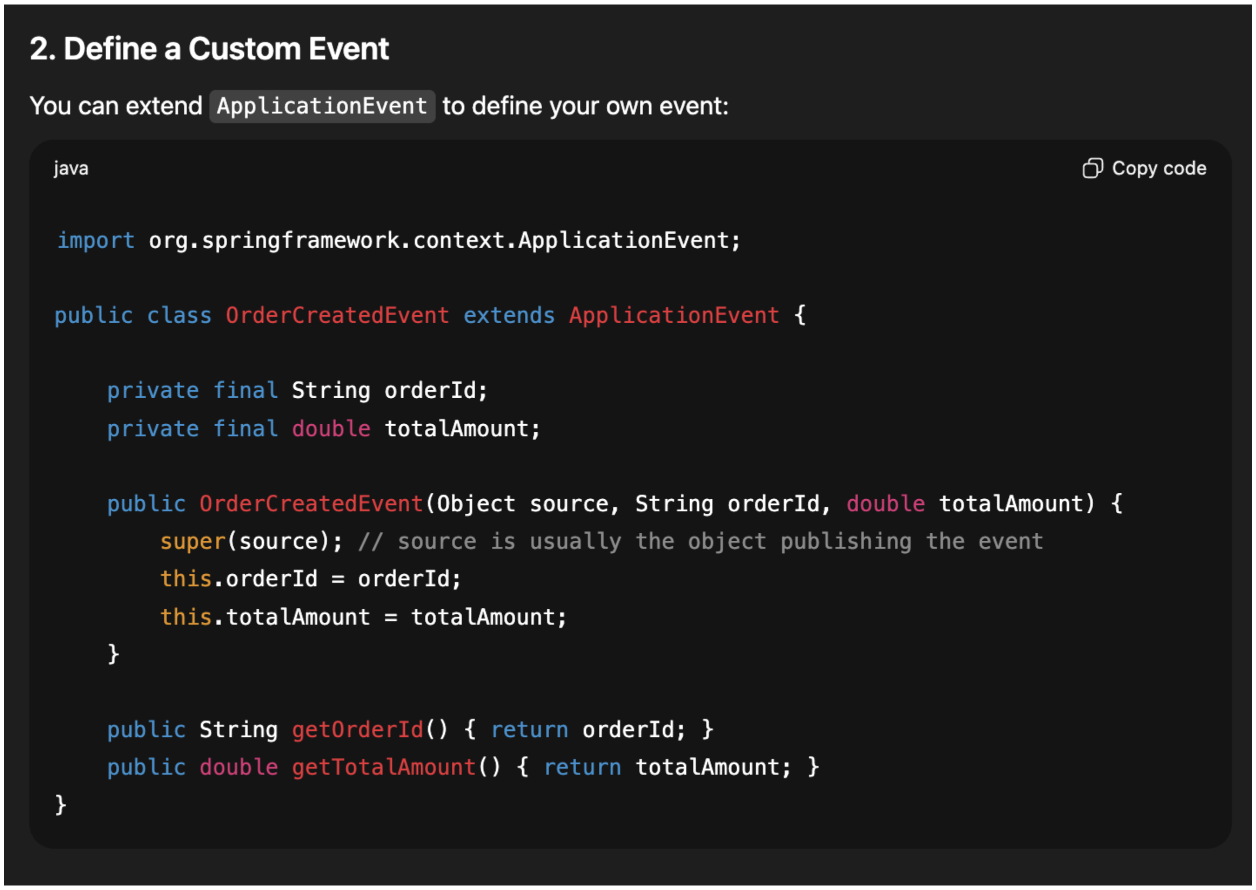
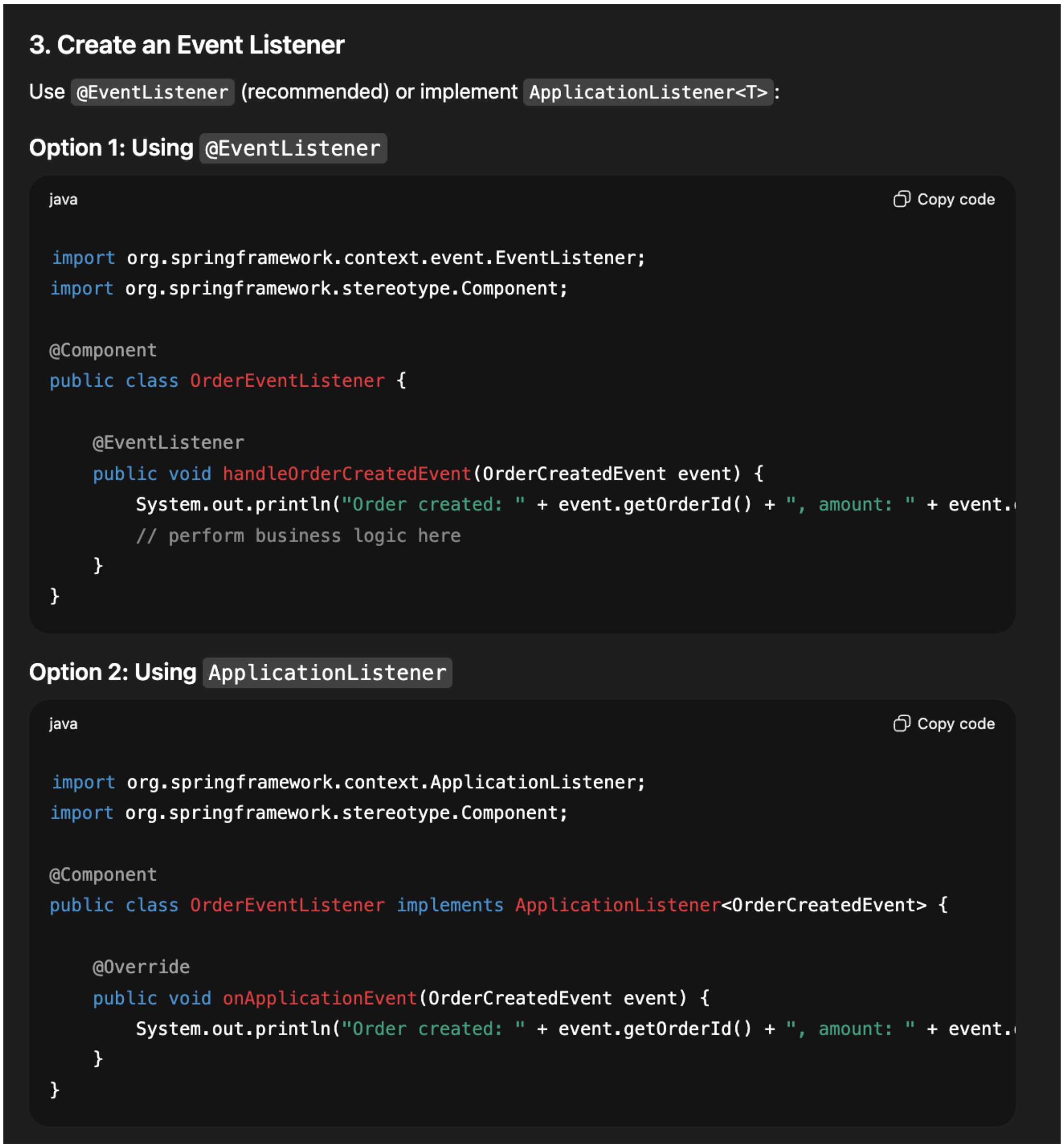
1. Application Event



2. Define a Custom Event

You can extend `ApplicationEvent` to define your own event:

```java
import org.springframework.context.ApplicationEvent;

public class OrderCreatedEvent extends ApplicationEvent {

    private final String orderId;
    private final double totalAmount;

    public OrderCreatedEvent(Object source, String orderId, double totalAmount) {
        super(source); // source is usually the object publishing the event
        this.orderId = orderId;
        this.totalAmount = totalAmount;
    }

    public String getOrderId() { return orderId; }
    public double getTotalAmount() { return totalAmount; }
}
```


3. Create an Event Listener

Use `@EventListener` (recommended) or implement `ApplicationListener<T>`:

Option 1: Using `@EventListener`

java Copy code

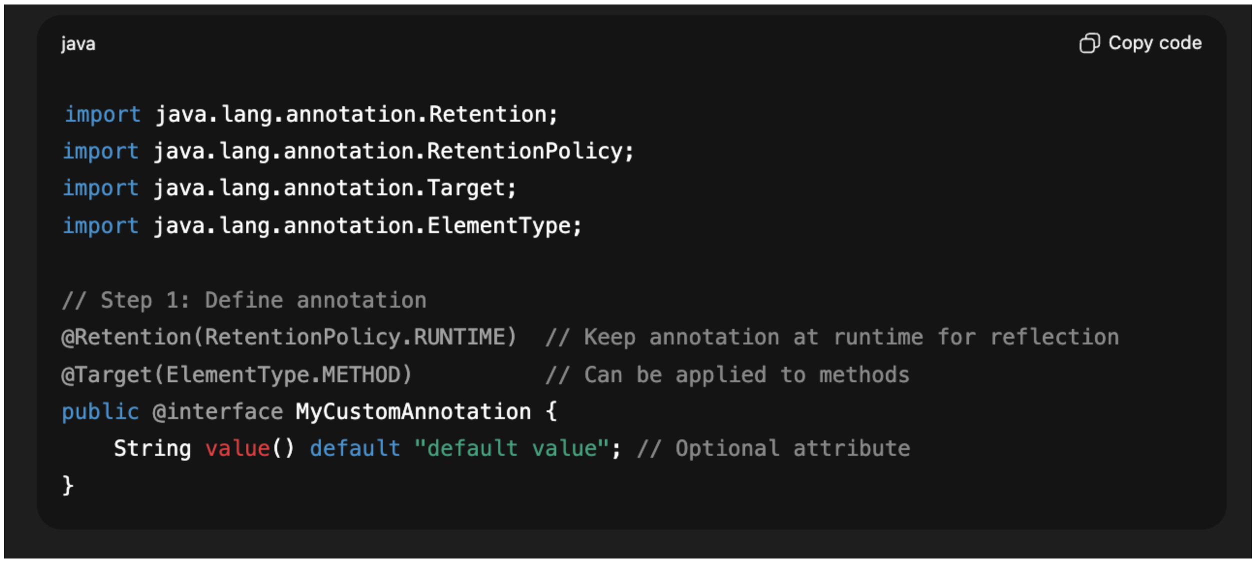
```java
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;

@Component
public class OrderEventListener {

    @EventListener
    public void handleOrderCreatedEvent(OrderCreatedEvent event) {
        System.out.println("Order created: " + event.getOrderId() + ", amount: " + event.
        // perform business logic here
    }
}
```

Option 2: Using `ApplicationListener`

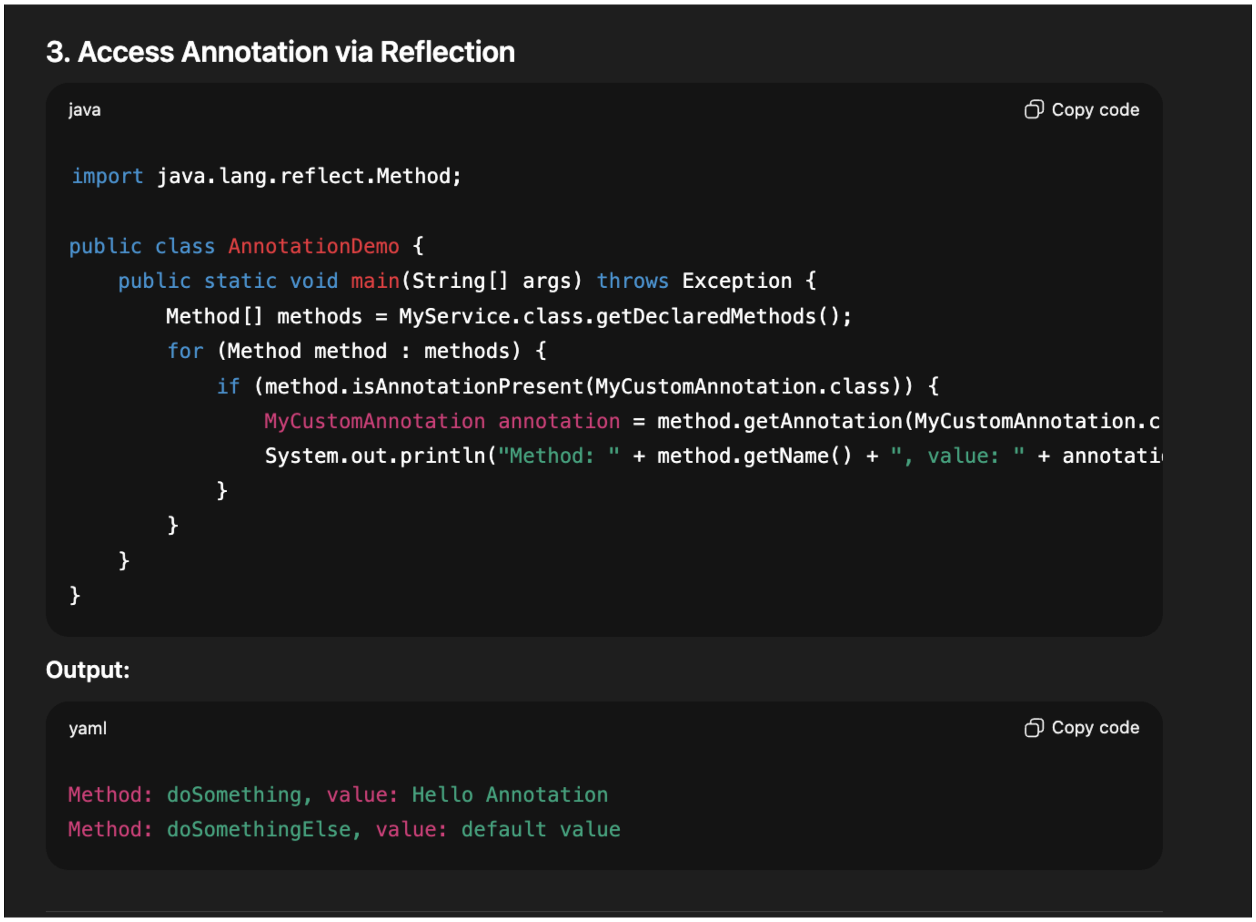
java Copy code

```java
import org.springframework.context.ApplicationListener;
import org.springframework.stereotype.Component;

@Component
public class OrderEventListener implements ApplicationListener<OrderCreatedEvent> {

    @Override
    public void onApplicationEvent(OrderCreatedEvent event) {
        System.out.println("Order created: " + event.getOrderId() + ", amount: " + event.
    }
}
```

2. Custom Annotations



```java
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;

// Step 1: Define annotation
@Retention(RetentionPolicy.RUNTIME)  // Keep annotation at runtime for reflection
@Target(ElementType.METHOD)          // Can be applied to methods
public @interface MyCustomAnnotation {
    String value() default "default value"; // Optional attribute
}
```

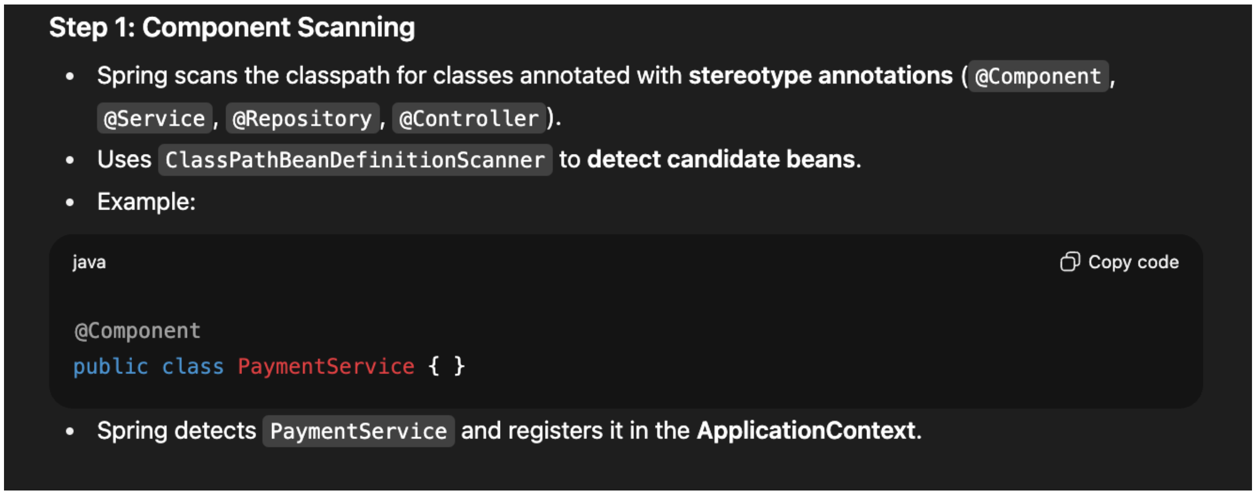


3. Access Annotation via Reflection

```java
import java.lang.reflect.Method;

public class AnnotationDemo {
    public static void main(String[] args) throws Exception {
        Method[] methods = MyService.class.getDeclaredMethods();
        for (Method method : methods) {
            if (method.isAnnotationPresent(MyCustomAnnotation.class)) {
                MyCustomAnnotation annotation = method.getAnnotation(MyCustomAnnotation.c
                System.out.println("Method: " + method.getName() + ", value: " + annotati
            }
        }
    }
}
```

Output:

```yaml
Method: doSomething, value: Hello Annotation
Method: doSomethingElse, value: default value
```



Step 1: Component Scanning

- Spring scans the classpath for classes annotated with **stereotype annotations** (`@Component`, `@Service`, `@Repository`, `@Controller`).
- Uses `ClassPathBeanDefinitionScanner` to **detect candidate beans**.
- Example:

java Copy code

```java
@Component
public class PaymentService { }
```

- Spring detects `PaymentService` and registers it in the **ApplicationContext**.