Hiding Information

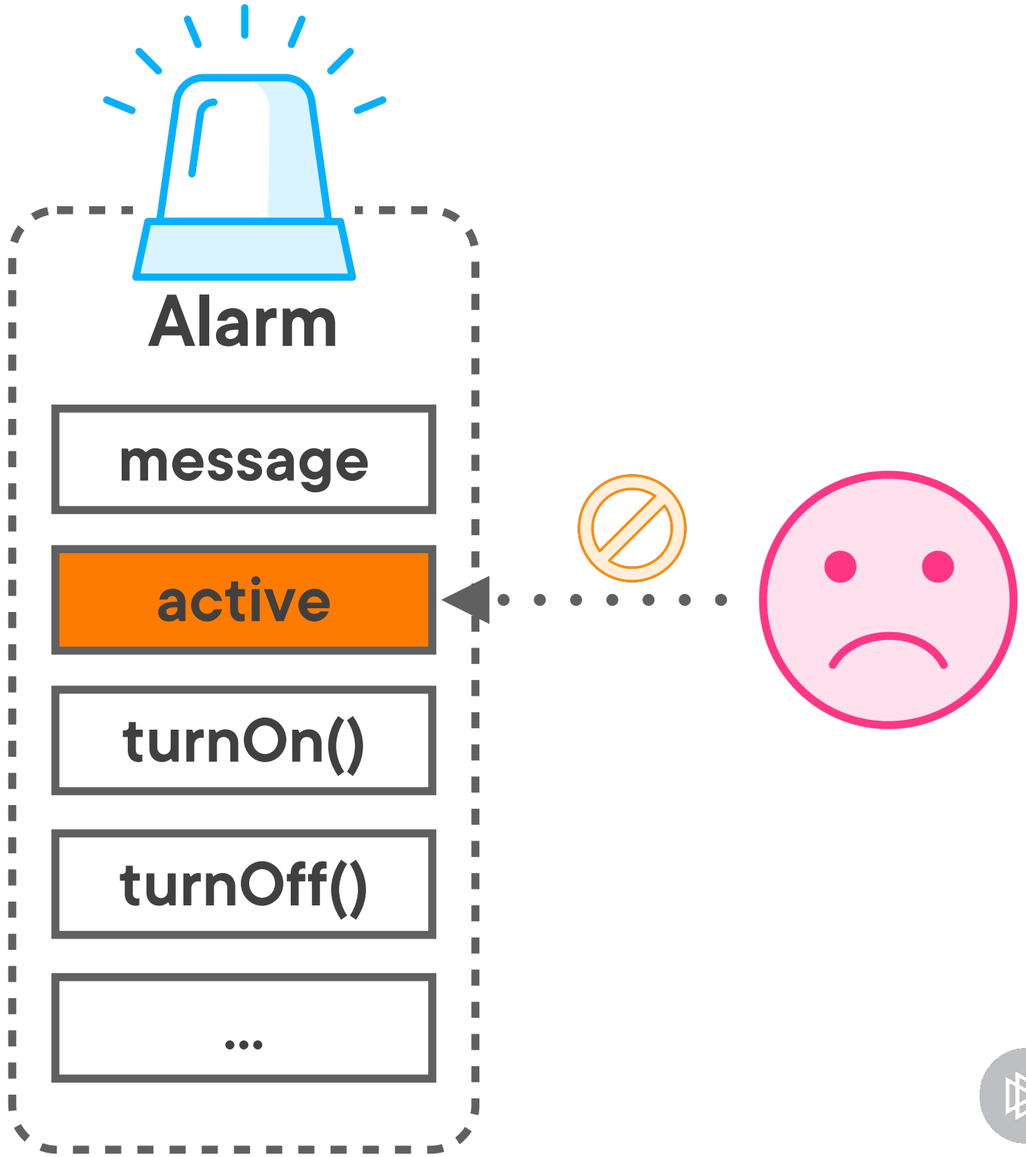
Paolo Perrotta

Developer, Author

@nusco | www.paoloperrotta.com

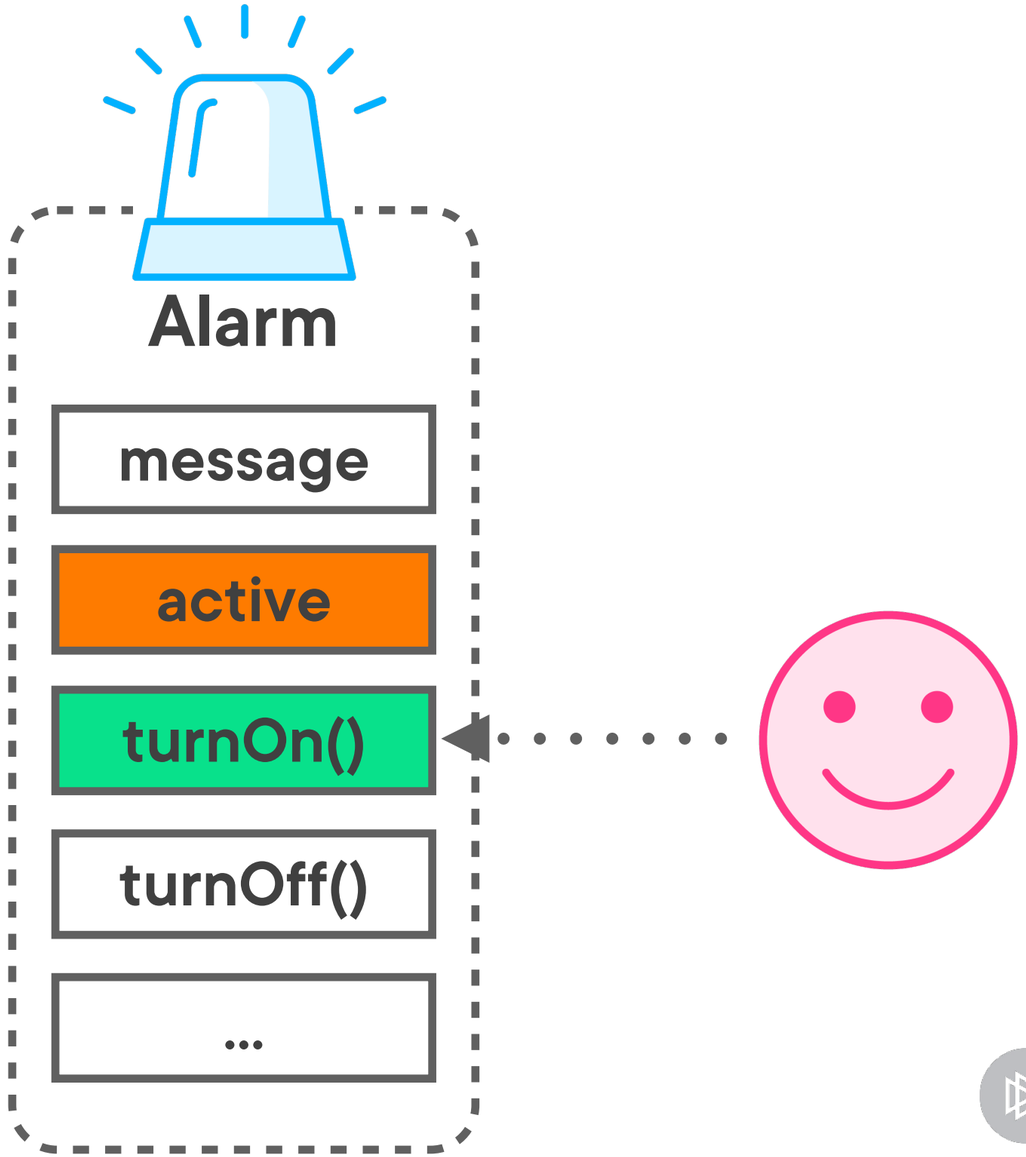
Hiding Fields and Methods

```
class Alarm {
    final String message;
    private boolean active;

    Alarm(String message) {
        this.message = message;
    }

    void turnOn() {
        active = true;
    }

    void turnOff() {
        active = false;
```



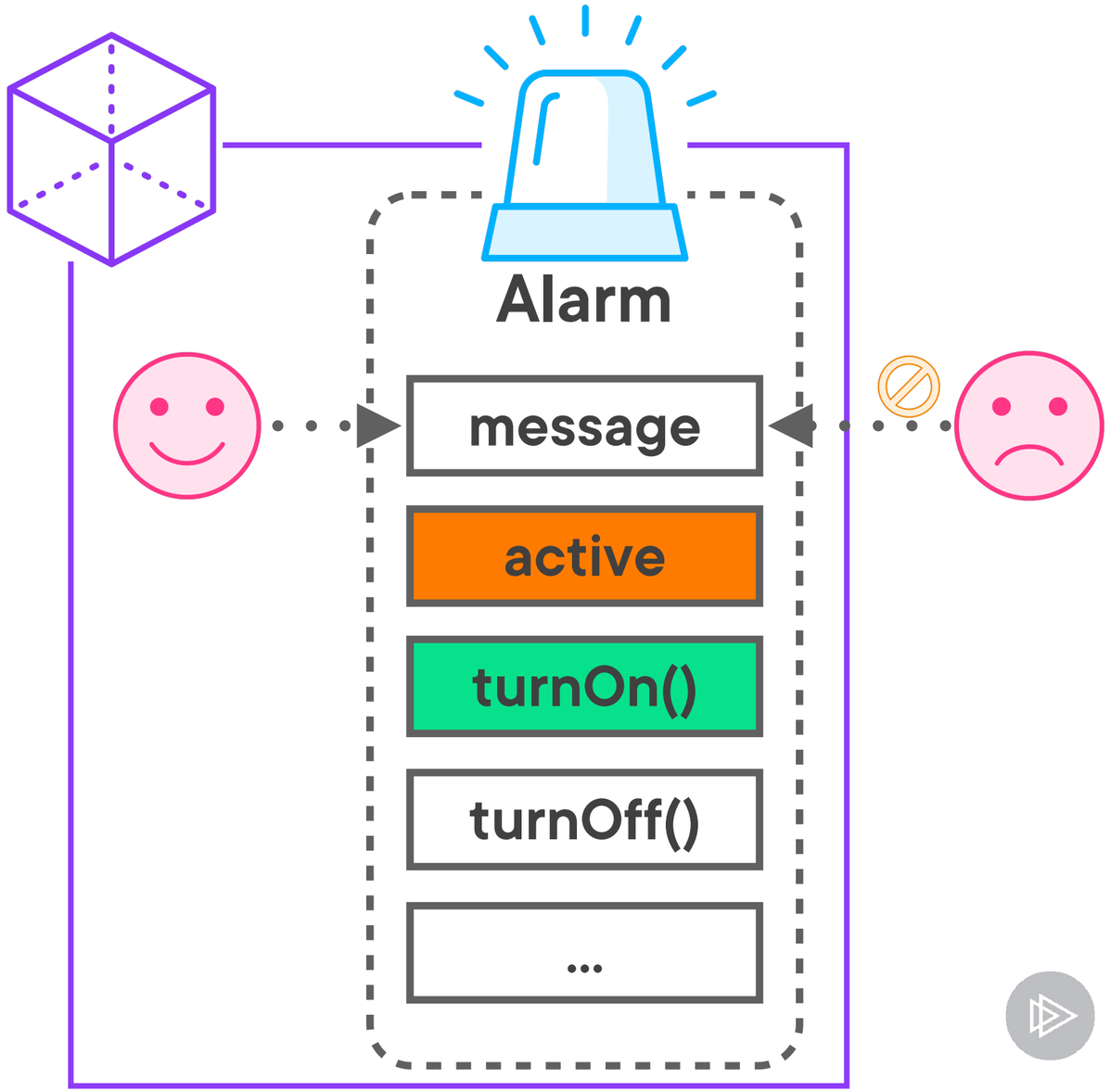
Hiding Fields and Methods

```
class Alarm {
    final String message;
    private boolean active;

    Alarm(String message) {
        this.message = message;
    }

    public void turnOn() {
        active = true;
    }

    void turnOff() {
        active = false;
```



“Package Private”

```
class Alarm {
    final String message;
    private boolean active;

    Alarm(String message) {
        this.message = message;
    }

    public void turnOn() {
        active = true;
    }

    void turnOff() {
        active = false;
```



Controlling Access to Classes

```
public class Alarm {
    final String message;
    private boolean active;

    Alarm(String message) {
        this.message = message;
    }

    public void turnOn() {
        active = true;
    }

    void turnOff() {
```

Controlling Access

A class can be:

- *public*: visible to code in other packages
- "package private": visible to code in its package

A field, method, or constructor can be:

- *public*: visible to code in other packages
- "package private": visible to code in its package
- *private*: only visible to code in the same class

More Ways to Control Access

protected

Another keyword like *public* and *private*.

Private Classes

Applies to “inner” (nested) classes.

Modules

Yet another way to control access.

The *Alarm* Class

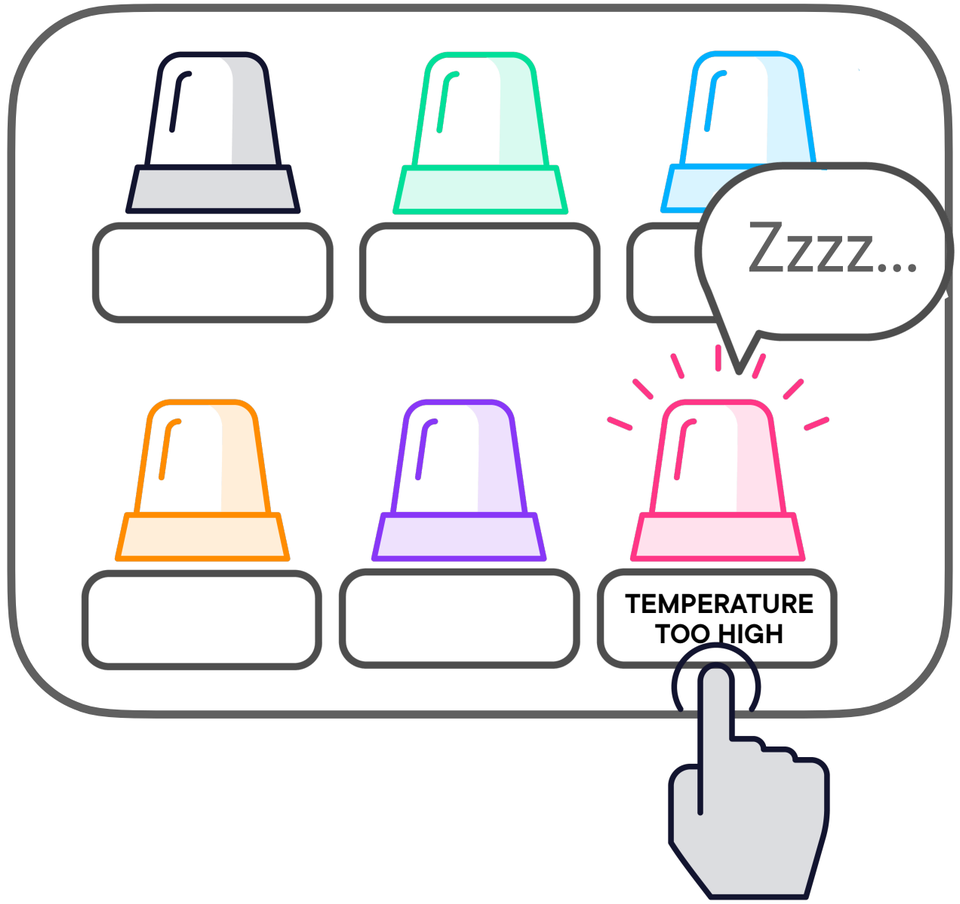
```
class Alarm {
    boolean active;
    final String message;

    Alarm(String message) {
        this.message = message;
    }

    void turnOn() {…}
    void turnOff() {…}
    String getReport() {…}
    String getReport(boolean uppercase) {…}
    void sendReport() {…}
}
```

A New Feature: Snoozing

You can snooze an alarm

- A snoozed alarm stays active, but getReport() returns an empty string
- Snoozing ends automatically after 5 minutes

Useful on a Control Panel



We Have a Problem

```
Alarm alarm = new Alarm("Temperature too high");
alarm.snooze();
alarm.turnOn();
```

We Have More Than One Problem

```
Alarm alarm = new Alarm("Temperature too high");
alarm.turnOn();
alarm.snooze();
alarm.turnOff();
alarm.turnOn();
```

Another Inconsistent Alarm

```
Alarm alarm = new Alarm("Temperature too high");
alarm.turnOn();
alarm.snooze();
alarm.active = false;
```

Fixed!

```
Alarm alarm = new Alarm("Temperature too high");
alarm.turnOn();
alarm.snooze();
alarm.active = false;
```

The *Alarm* Class, Updated

```
public class Alarm {
    private boolean active;
    private final String message;
    private LocalDateTime snoozeUntil;

    public Alarm(String message) {
        this.message = message;
        stopSnoozing();
    }

    public void snooze() {
        if (active)
            snoozeUntil = LocalDateTime.now().plusMinutes(5);
    }
```

The *Alarm* Class

Alarm

active	turnOn()
message	turnOff()
snoozeUntil	getReport()
snooze()	getReport(boolean)
isSnoozing()	sendReport()
getMessage()	stopSnoozing()

Interface and Implementation

Alarm

active

message

snoozeUntil

stopSnoozing()

turnOn()

turnOff()

getReport()

getReport(boolean)

sendReport()

snooze()

isSnoozing()

getMessage()

Thanks to Encapsulation, Classes Are...

...easier to use

You only care about the interface, not the implementation

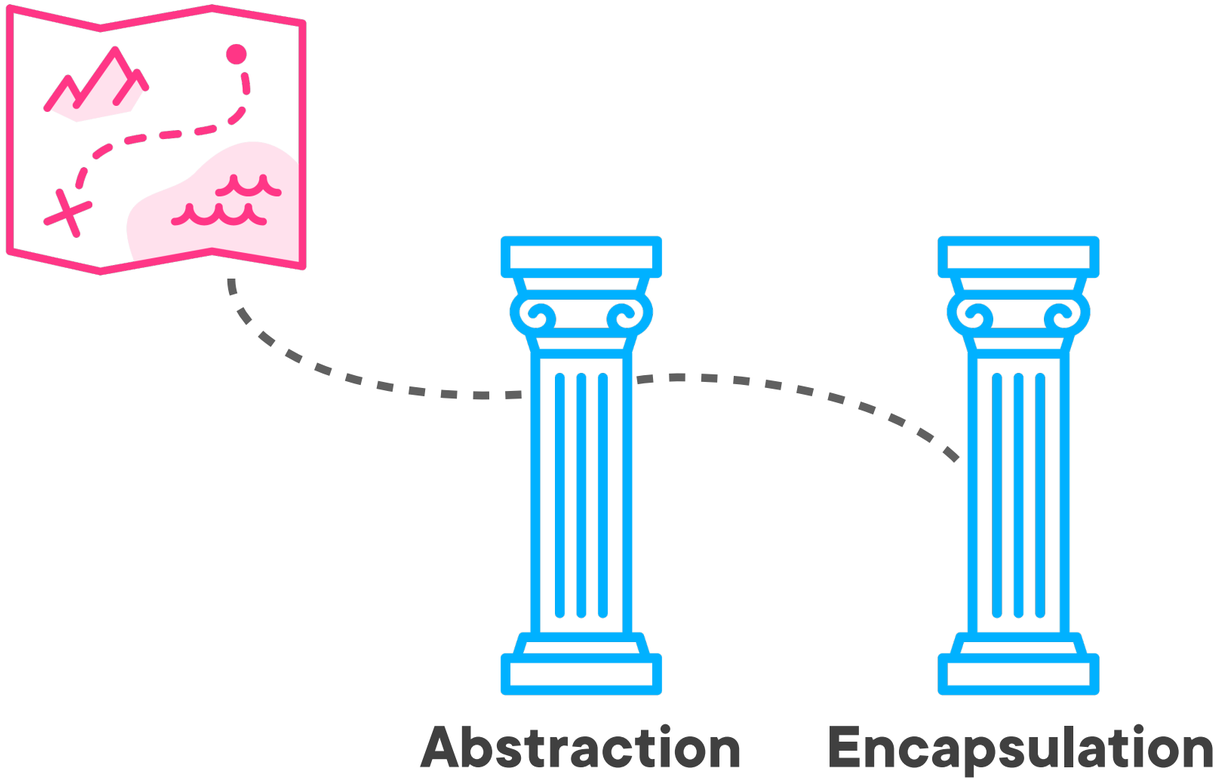
...harder to misuse

You protect objects from inconsistent changes

...easier to change

You can change the implementation without breaking clients

Encapsulation



Summary

Access modifiers: *public*, *private*, “package private”

Separating interface and implementations

The benefits of encapsulation

Up Next:

Designing with Abstraction and Encapsulation