Balking

Concurrency Decoupling About 2 min

Intent

Balking Pattern is used to prevent an object from executing a certain code if it is in an incomplete or inappropriate state.

Explanation

Real world example

> There's a start-button in a washing machine to initiate the laundry washing. When the washing machine is inactive the button works as expected, but if it's already washing the button does nothing.

In plain words

> Using the balking pattern, a certain code executes only if the object is in particular state.

Wikipedia says

> The balking pattern is a software design pattern that only executes an action on an object when the object is in a particular state. For example, if an object reads ZIP files and a calling method invokes a get method on the object when the ZIP file is not open, the object would "balk" at the request.

Programmatic Example

In this example implementation, `WashingMachine` is an object that has two states in which it can be: ENABLED and WASHING. If the machine is ENABLED, the state changes to WASHING using a thread-safe method. On the other hand, if it already has been washing and any other thread executes `wash()` it won't do that and returns without doing anything.

Here are the relevant parts of the `WashingMachine` class.

java

```java
@Slf4j
public class WashingMachine {

  private final DelayProvider delayProvider;
  private WashingMachineState washingMachineState;

  public WashingMachine(DelayProvider delayProvider) {
    this.delayProvider = delayProvider;
    this.washingMachineState =
WashingMachineState.ENABLED;
  }

  public WashingMachineState getWashingMachineState()
{
    return washingMachineState;
  }

  public void wash() {
    synchronized (this) {
      var machineState = getWashingMachineState();
      LOGGER.info("{}: Actual machine state: {}",
Thread.currentThread().getName(), machineState);
      if (this.washingMachineState ==
WashingMachineState.WASHING) {
        LOGGER.error("Cannot wash if the machine has
been already washing!");
        return;
      }
      this.washingMachineState =
WashingMachineState.WASHING;
    }
    LOGGER.info("{}: Doing the washing",
Thread.currentThread().getName());
    this.delayProvider.executeAfterDelay(50,
TimeUnit.MILLISECONDS, this::endOfWashing);
  }
  public synchronized void endOfWashing() {
    washingMachineState = WashingMachineState.ENABLED;
    LOGGER.info("{}: Washing completed.",
Thread.currentThread().getId());
  }
}
```

Here's the simple `DelayProvider` interface used by the `WashingMachine`.

```java
public interface DelayProvider {
  void executeAfterDelay(long interval, TimeUnit
timeUnit, Runnable task);
}
```

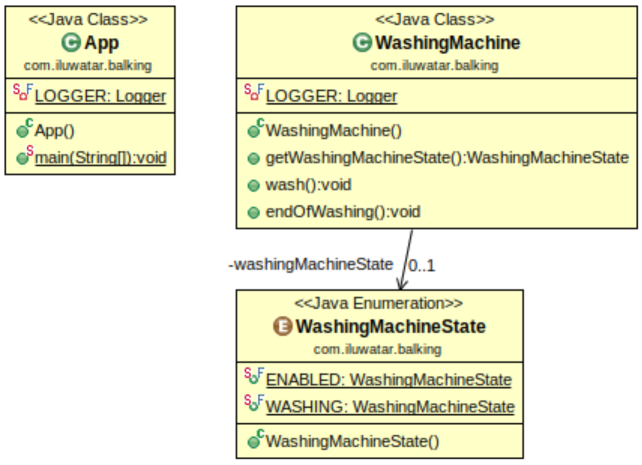
Now we introduce the application using the `WashingMachine`.

```java
  public static void main(String... args) {
    final var washingMachine = new WashingMachine();
    var executorService =
Executors.newFixedThreadPool(3);
    for (int i = 0; i < 3; i++) {
      executorService.execute(washingMachine::wash);
    }
    executorService.shutdown();
    try {
      executorService.awaitTermination(10,
TimeUnit.SECONDS);
    } catch (InterruptedException ie) {
      LOGGER.error("ERROR: Waiting on executor service
shutdown!");
      Thread.currentThread().interrupt();
    }
  }
```

Here is the console output of the program.

```text
14:02:52.268 [pool-1-thread-2] INFO com.iluwatar.balking.WashingMachine - pool-1-thread-2: Actual machine state: ENABLED
14:02:52.272 [pool-1-thread-2] INFO com.iluwatar.balking.WashingMachine - pool-1-thread-2: Doing the washing
14:02:52.272 [pool-1-thread-3] INFO com.iluwatar.balking.WashingMachine - pool-1-thread-3: Actual machine state: WASHING
14:02:52.273 [pool-1-thread-3] ERROR com.iluwatar.balking.WashingMachine - Cannot wash if the machine has been already washing!
14:02:52.273 [pool-1-thread-1] INFO com.iluwatar.balking.WashingMachine - pool-1-thread-1: Actual machine state: WASHING
14:02:52.273 [pool-1-thread-1] ERROR com.iluwatar.balking.WashingMachine - Cannot wash if the machine has been already washing!
14:02:52.324 [pool-1-thread-2] INFO com.iluwatar.balking.WashingMachine - 14: Washing completed.
```

Class diagram



Applicability

Use the Balking pattern when

- You want to invoke an action on an object only when it is in a particular state
- Objects are generally only in a state that is prone to balking temporarily but for an unknown amount of time

Related patterns

- Guarded Suspension Pattern (https://java-design-patterns.com/patterns/guarded-suspension/)
- Double Checked Locking Pattern (https://java-design-patterns.com/patterns/double-checked-locking/)

Credits

- Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, 2nd Edition, Volume 1 (https://www.amazon.com/gp/product/0471227293/ref=as_li_qf_asin_il_tl?ie=UTF8&tag=javadesignpat-20&creative=9325&linkCode=as2&creativeASIN=0471227293&linkId=0e39a59ffaab93fb476036fecb637b99)