Prof. Dr. Eric Dubuis

Exercise on Template Method

Introduction

The template method pattern provides a base class that encompasses a family of algorithms. It provides the common interface to its clients. However, variants of similar algorithms are provided by subclasses.

A Family of Buffers

A *buffer* is an object which allows to store and retrieve items. If a *bounded* buffer is full then *producers* cannot store more items until one or more *consumers* have retrieved at least one item. If any type of buffer is empty then consumers cannot consume any (non-existent!) item. A *one-place* bounded buffer allows to store one item only. (This type of buffer is only useful for this exercise.) An *unbounded* buffer allows to store (in principle) an unbounded number of items. A *bounded* buffer of size *n* allows to store up to *n* items at most.

The *public* interface of this family of buffers offers the following *synchronized* methods:

- public synchronized void put(E item) throws InterruptedException
  – If not full, adds an item at the end of the buffer, else blocks.
- public synchronized E get() throws InterruptedException
  – If not empty, removes an item at the head of the buffer, else blocks.
- public void init(int size) throws IllegalArgumentException
  – For bounded buffers only, allows to set the buffer size.

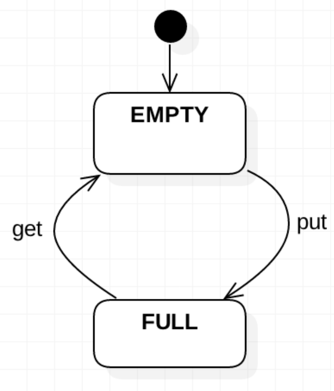
For each family member, the put() operation follows the following protocol:

1. Block until there is space available in the buffer. Method awaitPuttable() does this. Resume if there is space available. Then:
2. Store item in the buffer. Method store(Item x) does this. Then:
3. Wake up potentially sleeping consumers. Method checkState() does this.
4. Return.

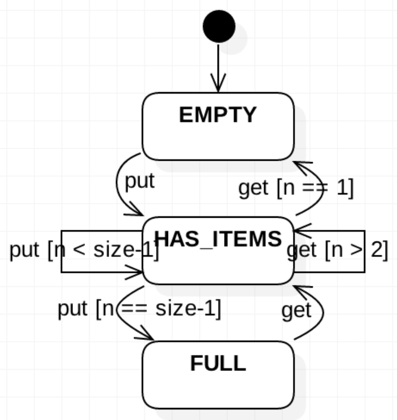
For each family member, the get() operation follows the following protocol:

1. Block until there is at least one item in the buffer. Method awaitGettable() does this. Resume if there is an item in the buffer. Then:
2. Retrieve the item in the head of the buffer. Method retrieve() does this.
3. Wake up potentially sleeping producers. Method checkState() does this.
4. Return.

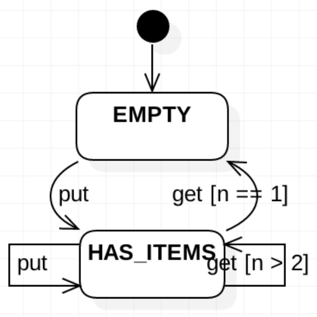
The behaviors of these three different types of buffers can each be sketched by the following corresponding state diagrams:



One-place buffer



Bounded buffer



Unbounded buffer

Task

Complete the abstract Buffer class as well as the concrete family member buffer classes you'll find in the source code. Watch the TODO markers. To test your implementation, run the JUnit tests (at the time being for class OnePlaceBuffer only).

Notice: The type of this project is Maven, i.e., this is a Maven project. Thus, import it as a Maven project into your IDE (Eclipse, InteliJ, ...).

Biel, December 12, 2017