BitVector

Overview

The goal of the first project is to implement a robust and configurable bit vector in Java 1.2. A `BitVector` maps integers in the active range of the vector to boolean values. A typical use for the class is to set a bit and to test for an individual bit. Bit vectors have a rich set of logical operations on pairs of vectors that include `and`, `or`, and `xor`.

The goal of this assignment is first and foremost to develop a correct implementation, but efficiency concerns can not be ignored as bit vectors may be used within efficiency critical portions of client programs.

Context

Although you are to implement a `BitVector` that is correct in all situations your clients are working on a project that has particular requirements and which are described in the two following use cases. You must strive to optimize your implementation for those cases.

Small-sets This scenario is concerned with small bits vectors for which the `test`, `or` and `and` methods are speed critical. Furthermore, the size of data structures has to minimized. In the small-set scenario, expect large numbers of vectors (thousands) with ranges from 0 to 127.

Big-sets This scenario has a number of large, sparse, bit vectors being used mostly for their set and test methods. In this scenario a space efficient representation of large vectors and speed of tests are critical. All other operations are used rather infrequently.

Example of applications of BitVectors include prime sieves and garbage collection (GC). In a GC one may represent a map of the memory by a bit vector to describe which objects are live and which ones are not (note that this is a gross simplification – we will see a more realistic representation later).

The `BitVector` interface is sufficiently general to allow for multiple implementations which can be optimized for the various uses of bit vectors. As a library implementer, you have to hide the complexity of these implementation behind a simple interface. You should approach this assignment as a software engineering task in which you have to produce robust and correct software. In particular, you should test your code thoroughly under the assumption that clients may execute arbitrary sequences of invocations of the methods exported in the interface. Your implementation must catch all errors and fail gracefully.

For simplicity, you can assume that client code is properly synchronized, in other words you don't need to be concerned about concurrency.

Interface Details

You are given the following interfaces *in lieu of* a specification:

- `BitVector` – represents one bit vector object.
- `Builder` – creates bit vectors of a particular kind.
- `Director` – creates a builder from a descriptor.
- `Descriptor` – specifies the requirements placed on a bit vector, these requirements include the active range, as well as some efficiency settings.

With the exception of `BitVector.Descriptor` all of the above are interfaces. Figure 1 illustrates some of the relationships between interfaces with a UML diagram. The Java code is contained in the two files `BitVector.java` and `Director.java`. This code uses inner interfaces to emphasize coupling between interfaces. `Director` has two inner interfaces `Director.Builder` and `Director.Descriptor` which stand for generic builders and descriptors. `BitVector` specializes the `Builder` interface and provides a concrete implementation for the `Descriptor`.

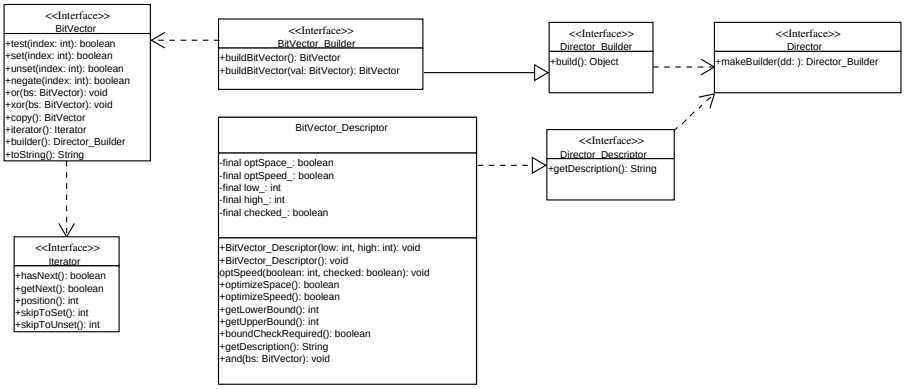

Figure 1: Class diagram for the PA1

To leave many customization opportunities open, we use the Builder pattern [GoF] for creating bit vectors. A simple example of the process is shown below, we assume that `MyDirector` is a class which implements the `Director` interface. First, we must perform some initialization. In an application that uses vectors of a single size this work can be done once and for all in global variables.

```
// Create one instance of a director.
static final Director director = new MyDirector();
// Create a descriptor for small and compact
// bit vectors with bound checks enabled.
static final BitVector.Descriptor small
              =  new BitVector.Descriptor(0, 31, false, true, true);
// Create a builder
static final BitVector.Builder builder
              = (BitVector.Builder) director.make(small);
```

Once this set up work is done all the code can simply use the `builder` field to construct new bit vectors:

```
void method(BitVector bv) {
    BitVector bs = builder.build();
    bs.set(15);
    if (bs.test(i))   bv.or(bs);
```

The key customization step comes in when `Director` is given a descriptor. At this step the director can choose a builder appropriate to the requirements embodied by the descriptor or throw an exception if these requirements can not be met. Potentially there can be many different classes which implement the `Builder` and `BitVector` interfaces. The role of the director is to pick the one most suited to the task at hand. Once an application is in possession of a builder no more customization is required or possible, as the only thing that a builder can do is to create new vectors.

Additional information along with code templates and complete JavaDoc can be found on the course web page.

Grading

This assignment will be graded with respect to the following criteria in decreasing order of importance: correctness, support for the use cases (i.e. optimized implementations for speed and space), quality of code and documentation, development log.

Requirements

For PA1 we expect you to turn in several Java classes. Please beware that you follow the naming convention. Failing to do so may prevent our grading programs from processing your submission. All `.java` files in your assignment should be named `ClassName_username.java` where `username` is your login. Thus `BitVector_jv.java` is a valid name. There must at least be one file named `Director_username.java` containing a `public` class of the same name, with a `public` constructor taking no arguments. All classes must be in the `cs307.first` package.

The assignment due date is next Sunday at 23:59 pm. The code should be submitted with "`turnin -c 307 -p pa1 filelist`". Please submit only the java files in the `first` directory.