CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2018
Doc 11 Assignment 2 Comments
Oct 11, 2018

Motivation Use Case for Priority Queue

Student waitlists

SDSU
- Over 6,000 classes each term
  - Each has a waitlist

~30,000 students
- A student can be in multiple waitlists

```
class PriorityQueue extends AbstractQueue {
    private Vector elements;
    private Vector undoStack;

    public Vector getUndoStack() {
        return undoStack;
    }
```

```
private Association<QueueObject, Integer> studentRecord
```

```
class PriorityQueue<E extends Student> extends AbstractQueue<E> {

class PriorityQueue<E extends Comparable> extends AbstractQueue<E> {

public class StrategyPriorityQueue<E> extends PriorityQueue<E>
```

public class priorityStrategy<E extends Student>

```
public class AddCommand implements Command {
    PriorityQueue pq;
    Stack<Student> history;

    public AddCommand(PriorityQueue pq) {
        this.pq = pq;
        this.history = new Stack<Student>():
    }
```

Why history?

```
public void execute( Student element) {
    pq.add( element);
    history.push(element);
}

public void undo() {
    pq.remove( history.pop());
}
```

```
public class AddCommand implements Command {
    PriorityQueue pq;
    Stack<Student> history;

    public AddCommand(PriorityQueue pq) {
        this.pq = pq;
        this.history = new Stack<Student>();
    }

public void execute( Student element) {
    pq.add( element);
    history.push(element);
}

public void undo() {
    pq.remove( history.pop());
}
```

```
Student a = new Student();
...
Student d = new Student();
PriorityQueue students = new Priority
AddCommand add =
    new AddCommand(students);
RemoveCommand remove =
    new RemoveCommand(students);
```

add(a);
add(c)
remove(c)
add(b)

How do we undo the operations?
Need to remember order

```
public class CommandInvoker {
    Command command;

    public CommandInvoker( Command command) {
        this.command = command;
    }
```

What do we gain by this class

```
public void execute(Student element) {
    command.execute(element);
}

public void undo() {
    command.undo();
}
```

```
public class AddCommand implements Command {
    PriorityQueue receiver;
    Student argument;

    public AddCommand(PriorityQueue queue, Student toAdd) {
        receiver = queue;
        argument = toAdd;
    }

    public void execute() {
        receiver.add(argument);
    }

    public void undo() {
        receiver.remove(argument);
    }
```

```
public class CommandInvoker {
    Stack<Command> doneCommands = new Stack<Command>();
    Stack<Command> undoneCommands = new Stack<Command>();

    public void execute(Command toDo) {
        toDo.execute();
        doneCommands.push(toDo);
    }

    public void undo() {
        Command toUndo = doneCommands.pop();
        toUndo.undo();
        undoneCommands.push(toUndo):
    }

    public void redo() {
        Command toRedo = undoneCommands.pop();
        toRedo.execute();
        doneCommands.push(toRedo):
    }
}
```

```
public class AddCommand<E> implements Command {
    List<E> elements;

    public AddCommand(E element) {
        List<E> convertedList = new ArrayList<>();
        convertedList.add(element);
        elements = convertedList;
    }

    public void execute(Queue queue) {
        for (E element: elements) {
            queue.add(element);
        }
    }

    public void undo(Queue queue) {
        for (E element: elements) {
            queue.remove(element);
        }
    }
```

```
public AddCommand(E element) {
    elements = new ArrayList<>();
    elements.add(element);
}
```

With 6,000+ queues
How do we know which one
to pass to undo?

How do we handle mixture of
AddCommands and Remove Commands

```
public interface Iterator extends java.util.Iterator<T> {
    public boolean hasNext();
    public T next();
}
```

```
public interface java.util.Iterator<T> {
    default void forEachRemaining(Consumer<? super E> action
    public boolean hasNext();
    public T next();
    default void remove()
}
```

```
public class StrategyContext<T> {
    IStrategy<T> strategy;

    public StrategyContext(IStrategy<T> strategy) {
        this.strategy = strategy;
    }

    public double prioritize(T object) {
        return strategy.prioritize(object);
    }
}
```

```
public void undoPriorityQueueAction() {
    Command undoCommand = getLastCommand();

    if (undoCommand instanceof AddCommand) {
        ((AddCommand) undoCommand).undoexecute();
    }
    if (undoCommand instanceof RemoveCommand) {
        ((RemoveCommand) undoCommand).undoexecute();
    }
}
```

```
public void undoPriorityQueueAction() {
    Command undoCommand = getLastCommand();
    undoCommand.undoexecute();
}
```

Avoid using instanceof

Replace instanceof with polymorphism

How Many Iterators at Time?

```
class PriorityQueue<T> {
    var elements: [T]
    var iteratorIndex: Int?
```

Is it reset when creating iterator

Since the queue hold the location of the iterator only one should exist at a time

But no such constraint exists

```
class PriorityQueue extends AbstractQueue {
        blah blah
    }

class Iterator extends PriorityQueue {

}
```

```
class PriorityQueue(Queue):
    def __init__(self,maxSize):
        self.queue = [ ]
        self.__priority_strategy = PriorityClass.Priority(PriorityClass.default)

        def add_to_priority_queue(self, data):
            blah

        def add(self, data):
            blah

    def _put(self, data):
        blah

    def get_priority_queue(self, data):
        return self.queue
```

```
class PriorityQueue(Queue):

    def create_iterator(self):
        return QueueIterator(self)


    def __iter__(self):
        return QueueIterator(self)
```

```
class Command(object):

    def __init__(self, obj, index=0, item=None):
        self._pq = obj
        self._index = index
        self._item = item
```

```
class AddCommand(Command):
    def execute(self):
        self._index = self._pq._put(self._item)

    def undo(self):
        self._pq._get(self._index)
```