Mutator Encyclopedia Article

Mutator

The following sections of this BookRags Literature Study Guide is offprint from Gale's For Students Series: Presenting Analysis, Context, and Criticism on Commonly Studied Works: Introduction, Author Biography, Plot Summary, Characters, Themes, Style, Historical Context, Critical Overview, Criticism and Critical Essays, Media Adaptations, Topics for Further Study, Compare & Contrast, What Do I Read Next?, For Further Study, and Sources.

The following sections, if they exist, are offprint from Beacham's Encyclopedia of Popular Fiction: "Social Concerns", "Thematic Overview", "Techniques", "Literary Precedents", "Key Questions", "Related Titles", "Adaptations", "Related Web Sites".

The following sections, if they exist, are offprint from Beacham's Guide to Literature for Young Adults: "About the Author", "Overview", "Setting", "Literary Qualities", "Social Sensitivity", "Topics for Discussion", "Ideas for Reports and Papers".

Contents

Mutator

A mutator is a **function** or **method** that changes the state of the **object** it belongs to; this is in contrast to an "accessor," which only "reads" the state of the object it belongs to.

In object-oriented programs, the program's **data** typically comprises "objects" that represent things or concepts in the real world. Objects not only have data, sometimes called "state" or "identity"; but they also have behavior, or things that they can do, either on their own data or data given to them by other objects. An object that represents an egg-timer might contain data about how long to boil an egg, and it might have a function to ring a bell once the egg has been boiling for long enough.

When a function that is associated with an object actually changes some of the data in the object, the function is referred to as a "mutator function" because it quite literally "mutates" the object into a different state.

As an example, consider a simple **C++** eggTimer class:

- class eggTimer {
- public:
- eggTimer() : _hasBeenSet(false), _timeVal(0) {} // constructor
- void setTimer(int t) { _timeVal = t;} // mutator
- int getTimer() const { _hasBeenSet = true; return _timeVal;} // accessor
- private:
- int _timeVal;
- mutable bool _hasBeenSet; // can be changed in a const function
- };

In the above **code**, the function setTimer() is a mutator function because it changes the data in the eggTimer object by setting its _timeVal member to a given **value**. But the setTimer() function is an accessor function because it does not change anything about the object.

To make the **compiler** enforce this, getTimer() has been declared as "const," which means that if a programmer mistakenly did something in writing the function that did change the object, the compiler would detect this and issue an error.

The interesting thing to note is that even though getTimer() has been declared "const," the member **variable** _hasBeenSet is being changed. This is allowed because _hasBeenSet has been declared as "mutable," which tells the compiler that it is allowed to change the value of _hasBeenSet in a "const" function.