The Singleton Pattern

Motivating Example

Douglas C. Schmidt
Learning Objectives in This Lesson

- Recognize how the *Singleton* pattern can be applied to centralize access to global resources in the expression tree processing app.
Learning Objectives in This Lesson

• Recognize how the *Singleton* pattern can be applied to centralize access to global resources.

[Diagram showing the Reactor, ET_Event_Handler, Verbose_Mode, and Succinct_Mode structures with associated methods like run_event_loop, handle_input, prompt_user, receive_input, make_command, and execute_command.]

See [en.wikipedia.org/wiki/Robot_B-9](en.wikipedia.org/wiki/Robot_B-9)
Motivating the Need for the Singleton Pattern in the Expression Tree App
A Pattern for Centralizing Global Resource Access

**Purpose:** Simplify access to global resources without using global variables.

The *Singleton* pattern has well-known drawbacks, so apply it with care.
int main (int argc, char *argv[]) {
    unique_ptr<Options> options
        (Options::instance ());

    if (!options->parse_args (argc, argv))
        return 0;

    unique_ptr<Reactor> reactor
        (Reactor::instance ());

    reactor->register_input_handler
        (ET_Event_Handler::make_handler
            (options->verbose ()));

    reactor->run_event_loop ();

    return 0;
}
Only one instance of certain classes are needed, e.g.,

- Command-line options that determine the app operating mode

```cpp
int main (int argc, char *argv[]) {
    unique_ptr<Options> options
        (Options::instance ());

    if (!options->parse_args (argc, argv))
        return 0;

    unique_ptr<Reactor> reactor
        (Reactor::instance ());

    reactor->register_input_handler
        (ET_Event_Handler::make_handler
            (options->verbose ()));

    reactor->run_event_loop () ;

    return 0;
}
```
Only one instance of certain classes are needed, e.g.,

- Command-line options that determine the app operating mode
- Reactor that drives the expression tree input handling

```cpp
int main (int argc, char *argv[]) {
    unique_ptr<Options> options (Options::instance ());

    if (!options->parse_args (argc, argv))
        return 0;

    unique_ptr<Reactor> reactor (Reactor::instance ());

    reactor->register_input_handler
        (ET_Event_Handler::make_handler
            (options->verbose ()));

    reactor->run_event_loop ();

    return 0;
}
```
Only one instance of certain classes are needed, e.g.,
- Command-line options that determine the app operating mode
- Reactor that drives the expression tree input handling

```cpp
int main (int argc, char *argv[]) {
    unique_ptr<Options> options
        (Options::instance ());

    if (!options->parse_args (argc, argv))
        return 0;

    unique_ptr<Reactor> reactor
        (Reactor::instance ());

    reactor->register_input_handler
        (ET_Event_Handler::make_handler
            (options->verbose ()));

    reactor->run_event_loop ();

    return 0;
}
```

Passing these objects as parameters can become tedious & “cluttered.”
Problem: Minimizing Global Variable Liabilities

- Global variables are problematic for several reasons.
  - Increase implicit dependencies & reduce program clarity
  - Incur time/space overhead even if they aren’t used
  - Cannot be extended transparently
  - May not be initialized & destroyed properly in certain programming languages & runtime environments

See [wiki.c2.com/?GlobalVariablesConsideredHarmful](http://wiki.c2.com/?GlobalVariablesConsideredHarmful)
Global variables are problematic for several reasons:

- Increase implicit dependencies & reduce program clarity
- Incur time/space overhead even if they aren’t used
- Cannot be extended transparently
- May not be initialized & destroyed properly in certain programming languages & runtime environments

This discussion wouldn’t address all liabilities with global variables.
Solution: Centralize Access to Global Resources

- Create a central access point to global resources \textit{without} using a global variable.

```cpp
int main (int argc, char *argv[]) {

    unique_ptr<Options> options(Options::instance ());

    if (!options->parse_args (argc, argv))
        return 0;

    unique_ptr<Reactor> reactor(Reactor::instance ());

    reactor->register_input_handler
        (ET_Event_Handler::make_handler(options->verbose ()));

    reactor->run_event_loop ();

    return 0;
}
```
Solution: Centralize Access to Global Resources

- Create a central access point to global resources *without* using a global variable.

```cpp
#include <iostream>
#include <memory>

using namespace std;

unique_ptr<Options> options(Options::instance());

if (!options->parse_args(argc, argv))
  return 0;

unique_ptr<Reactor> reactor(Reactor::instance());

reactor->register_input_handler
  (ET_Event_Handler::make_handler(options->verbose()));

reactor->run_event_loop();

return 0;
```

Allocate object on demand & parse common-line options
Solution: Centralize Access to Global Resources
• Create a central access point to global resources \textit{without} using a global variable.

```cpp
int main (int argc, char *argv[]) {

unique_ptr<Options> options(Options::instance());

if (!options->parse_args(argc, argv))
    return 0;

unique_ptr<Reactor> reactor(Reactor::instance());

reactor->register_input_handler
    (ET_Event_Handler::make_handler(options->verbose()));

reactor->run_event_loop();

return 0;
}
```
Solution: Centralize Access to Global Resources

- Create a central access point to global resources *without* using a global variable.

```cpp
int main (int argc, char *argv[]) {

    unique_ptr<Options> options(Options::instance ());

    if (!options->parse_args (argc, argv))
        return 0;

    unique_ptr<Reactor> reactor(Reactor::instance ());

    reactor->register_input_handler
        (ET_Event_Handler::make_handler(options->verbose ()�));

    reactor->run_event_loop ();

    return 0;
}
```

Allocate/register requested ET_Event_Handler based on command-line options
Solution: Centralize Access to Global Resources

- Create a central access point to global resources without using a global variable.

```cpp
int main (int argc, char *argv[]) {

    unique_ptr<Options> options(Options::instance());

    if (!options->parse_args (argc, argv))
        return 0;

    unique_ptr<Reactor> reactor(Reactor::instance());

    reactor->register_input_handler
        (ET_Event_Handler::make_handler(options->verbose()));

    reactor->run_event_loop();

    return 0;
}
```

Reactors process user input via callbacks