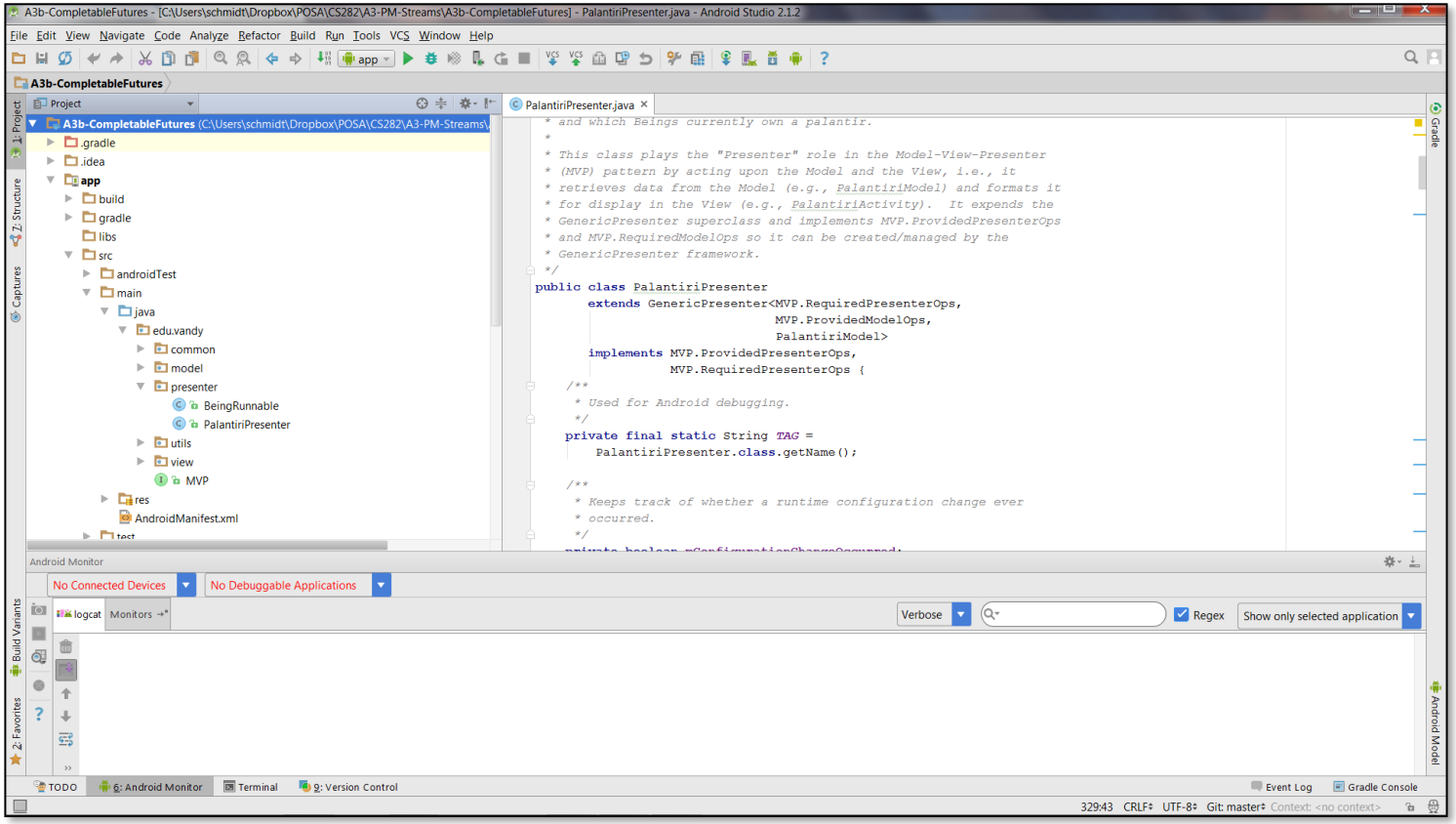
The Strategy Pattern

Implementation in C++

Douglas C. Schmidt

Learning Objectives in This Lesson

- Recognize how the *Strategy* pattern can be applied in the expression tree processing app to encapsulate variability of algorithm & platform behaviors via common APIs.
- Understand the structure & functionality of the *Strategy* pattern.
- Know how to implement the *Strategy* pattern in C++.



Strategy example in C++

- The **`begin()`** & **`end()`** factory methods in the **`Expression_Tree`** class returns the requested iterator strategy.

```
class Expression_Tree {
  ...
  iterator begin (const std::string &traversal_order) {
    return iterator(tree_iterator_factory.make_iterator
                        (*this, traversal_order, false));
  }

  iterator end (const std::string &traversal_order) {
    return iterator(tree_iterator_factory.make_iterator
                        (*this, traversal_order, true));
  }
}
```

Strategy example in C++

- The **`begin()`** & **`end()`** factory methods in the **`Expression_Tree`** class returns the requested iterator strategy.

```
class Expression_Tree {
  ...
  iterator begin (const std::string &traversal_order) {
    return iterator(tree_iterator_factory.make_iterator
                    (*this, traversal_order, false));
  }

  iterator end (const std::string &traversal_order) {
    return iterator(tree_iterator_factory.make_iterator
                    (*this, traversal_order, true));
  }
}
```

These factory methods forward to an internal factory.

Strategy example in C++

- The **`begin()`** & **`end()`** factory methods in the **`Expression_Tree`** class returns the requested iterator strategy.

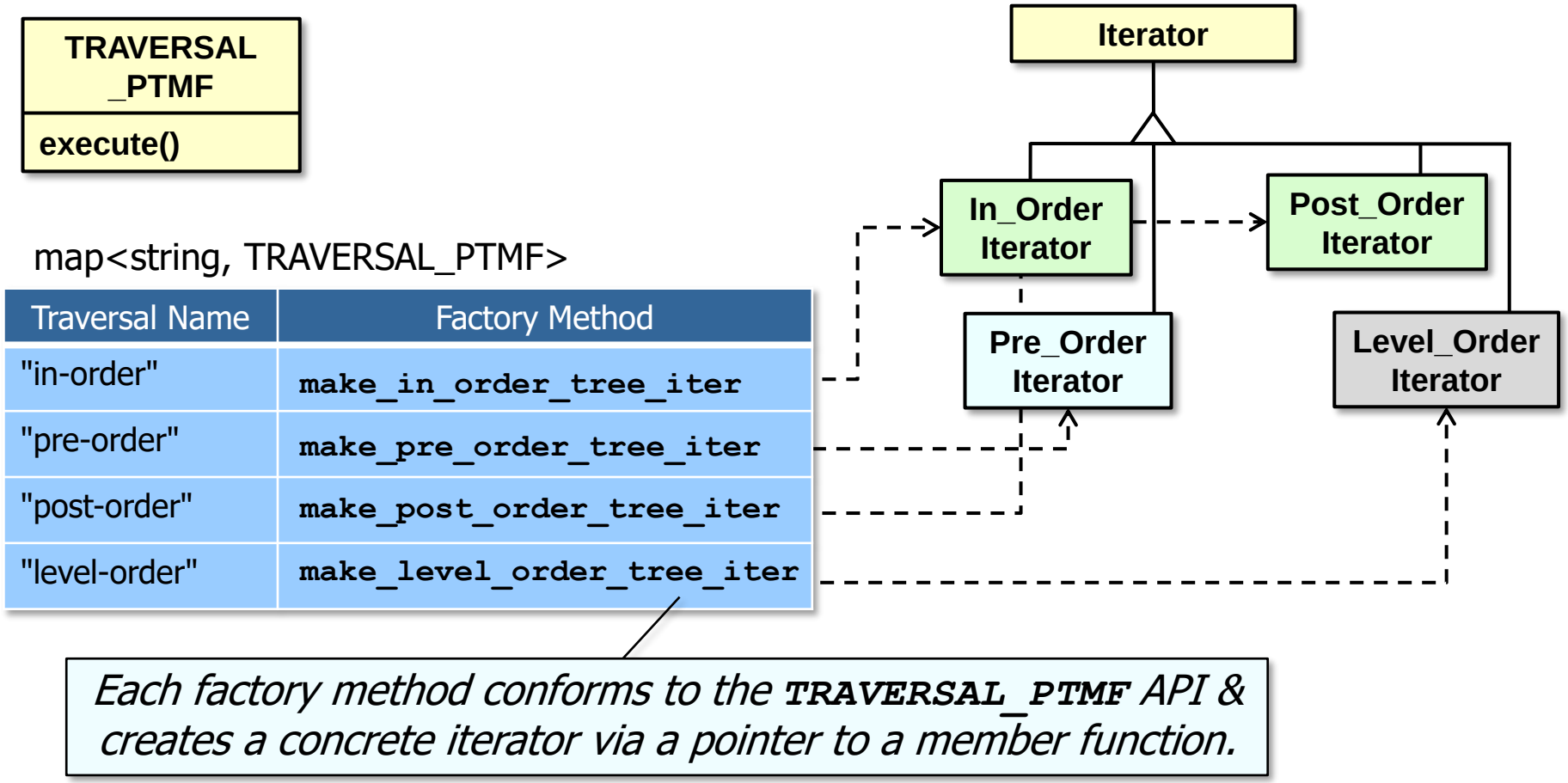
```
class Expression_Tree {
  ...
  iterator begin (const std::string &traversal_order) {
    return iterator(tree_iterator_factory.make_iterator
                    (*this, traversal_order, false));
  }

  iterator end (const std::string &traversal_order) {
    return iterator(tree_iterator_factory.make_iterator
                    (*this, traversal_order, true));
  }
}
```

This application of *Factory Method* is accessed via *Bridge* & implemented via *Command* to create the *Strategy* used as an *Iterator* to access each node in the *Composite* expression tree.

This OO design & implementation exhibits high-pattern density!

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.



map<string, TRAVERSAL_PTMF>

Traversal Name	Factory Method
"in-order"	`make_in_order_tree_iter`
"pre-order"	`make_pre_order_tree_iter`
"post-order"	`make_post_order_tree_iter`
"level-order"	`make_level_order_tree_iter`

*Each factory method conforms to the **`TRAVERSAL_PTMF`** API & creates a concrete iterator via a pointer to a member function.*

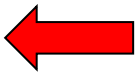
Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.

```
class ET_Iterator_Factory {

  typedef ET_Iter_Impl *(*TRAVERSAL_PTMF)(Expression_Tree &tree,
                                          bool end_iter);
  typedef std::map <std::string, TRAVERSAL_PTMF> TRAVERSAL_MAP;

  TRAVERSAL_MAP traversal_map_;


  ET_Iterator_Factory() {
    traversal_map_["in-order"] = &make_in_order_tree_iterator;
    traversal_map_["pre-order"] = &make_pre_order_tree_iterator;
    traversal_map_["post-order"] = &make_post_order_tree_iterator;
    ...
  }
}
```

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.



We apply *Command* to initialize `ET_Iterator_Factory`.

```
class ET_Iterator_Factory {

  typedef ET_Iter_Impl *(*TRAVERSAL_PTMF)(Expression_Tree &tree,
                                          bool end_iter);
  typedef std::map <std::string, TRAVERSAL_PTMF> TRAVERSAL_MAP;

  TRAVERSAL_MAP traversal_map_;


  ET_Iterator_Factory() {
    traversal_map_["in-order"] = &make_in_order_tree_iterator;
    traversal_map_["pre-order"] = &make_pre_order_tree_iterator;
    traversal_map_["post-order"] = &make_post_order_tree_iterator;
    ...
  }
}
```

Strategy example in C++

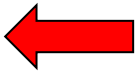
- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.


```
class ET_Iterator_Factory {

  typedef ET_Iter_Impl *(*TRAVERSAL_PTMF)(Expression_Tree &tree,
                                          bool end_iter);
  typedef std::map <std::string, TRAVERSAL_PTMF> TRAVERSAL_MAP;

  TRAVERSAL_MAP traversal_map_;


  ET_Iterator_Factory() {
    traversal_map_["in-order"] = &make_in_order_tree_iterator;
    traversal_map_["pre-order"] = &make_pre_order_tree_iterator;
    traversal_map_["post-order"] = &make_post_order_tree_iterator;
    ...
  }
}
```

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.

```
class ET_Iterator_Factory {

  typedef ET_Iter_Impl *(*TRAVERSAL_PTMF)(Expression_Tree &tree,
                                          bool end_iter);
  typedef std::map <std::string, TRAVERSAL_PTMF> TRAVERSAL_MAP;

  TRAVERSAL_MAP traversal_map_;

  ET_Iterator_Factory() {
    traversal_map_["in-order"] = &make_in_order_tree_iterator;
    traversal_map_["pre-order"] = &make_pre_order_tree_iterator;
    traversal_map_["post-order"] = &make_post_order_tree_iterator;
    ...
  }
}
```



Map strings to factory commands that create iterator strategies

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.

```
class ET_Iterator_Factory {

  typedef ET_Iter_Impl *(*TRAVERSAL_PTMF)(Expression_Tree &tree,
                                          bool end_iter);
  typedef std::map <std::string, TRAVERSAL_PTMF> TRAVERSAL_MAP;

  TRAVERSAL_MAP traversal_map_;

  ET_Iterator_Factory() {
    traversal_map_["in-order"] = &make_in_order_tree_iterator;
    traversal_map_["pre-order"] = &make_pre_order_tree_iterator;
    traversal_map_["post-order"] = &make_post_order_tree_iterator;
    ...
  }
}
```

Pointers-to-member-functions used to create iterator strategies

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.

The factory method

```
class ET_Iterator_Factory {

  ET_Iter_Impl *make_iterator (Expression_Tree &tree,
                               const std::string &order,
                               bool end_iter) {
  auto iter = traversal_map_.find (order);

  if (iter == traversal_map_.end ())
    throw Invalid_Iterator (order);


  else
    return (*iter->second) (tree, end_iter);
}
```

The factory method finds/executes the command that makes an iterator strategy.

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.

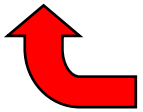
```
class ET_Iterator_Factory {

  ET_Iter_Impl *make_iterator (Expression_Tree &tree,
                               const std::string &order,
                               bool end_iter) {
  auto iter = traversal_map_.find (order);

  if (iter == traversal_map_.end ())
    throw Invalid_Iterator (order);


  else
    return (*iter->second) (tree, end_iter);
}
```

Try to find a pre-allocated factory command

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.

```
class ET_Iterator_Factory {

  ET_Iter_Impl *make_iterator (Expression_Tree &tree,
                               const std::string &order,
                               bool end_iter) {
  auto iter = traversal_map_.find (order);

  if (iter == traversal_map_.end ())
    throw Invalid_Iterator (order);


  else
    return (*iter->second) (tree, end_iter);
}
```



If found, execute it to make an iterator strategy

Strategy example in C++

- The **`ET_Iterator_Factory.make_iterator()`** factory method dynamically allocates the appropriate iterator strategy.

```
class ET_Iterator_Factory {

  ET_Iter_Impl *make_iterator (Expression_Tree &tree,
                               const std::string &order,
                               bool end_iter) {
  auto iter = traversal_map_.find (order);

  if (iter == traversal_map_.end ())
    throw Invalid_Iterator (order);

  else
    return (*iter->second) (tree, end_iter);
}
```

Otherwise, user gave unsupported request, so throw an exception