Intent

- Create families of related objects without specifying subclass names

Applicability

- When clients cannot anticipate groups of classes to instantiate

Structure
Abstract Factory example in C++

• Create families of related objects without specifying subclass names

```cpp
class Concrete_ET_Command_Factory_Impl
  : public ET_Command_Factory_Impl {
public:

  Concrete_ET_Command_Factory_Impl() {
    command_map_['format'] = &make_format_command;
    command_map_['expr'] = &make_expr_command;
    command_map_['eval'] = &make_eval_command;
    ...
  }
```

`std::map associating command names to pointer-to-member-function command factories`
Abstract Factory example in C++

• Create families of related objects without specifying subclass names

```cpp
class Concrete_ET_Command_Factory_Impl : public ET_Command_Factory_Impl {
public:
  Concrete_ET_Command_Factory_Impl() {
    command_map_['format'] = &make_format_command;
    command_map_['expr'] = &make_expr_command;
    command_map_['eval'] = &make_eval_command;
...
  }

  virtual ET_Command make_command(const std::string &input) {
    auto iter = command_map_.find(command_name(input));
    if (iter != command_map_.end()) {
      auto ptmf = iter->second;
      return (this->*ptmf)(command_parameter(input));
    }
  }
...
```

The primary factory method that creates the designated command based on user input

Dispatch command factory method via returned via map
Abstract Factory

Consequences

+ *Flexibility*: Removes type (i.e., subclass) dependencies from clients

+ *Abstraction & semantic checking*: Encapsulates product’s composition

- *Complexity*: Tedious to extend factory interface to create new products
Abstract Factory  

**Consequences**

+ *Flexibility*: Removes type (i.e., subclass) dependencies from clients
+ *Abstraction & semantic checking*: Encapsulates product’s composition

- *Complexity*: Tedious to extend factory interface to create new products

**Implementation**

- Parameterization as a way of controlling interface size
- Configuration with prototypes to determine who creates the factories
- Abstract factories are essentially groups of factory methods
Abstract Factory

Consequences

+ **Flexibility**: Removes type (i.e., subclass) dependencies from clients

+ **Abstraction & semantic checking**: Encapsulates product’s composition

- **Complexity**: Tedious to extend factory interface to create new products

Implementation

- Parameterization as a way of controlling interface size

- Configuration with prototypes to determine who creates the factories

- Abstract factories are essentially groups of factory methods

GoF Object Creational

Known Uses

- InterViews Kits
- ET++ WindowSystem
- AWT Toolkit
- The ACE ORB (TAO)
Summary of Command & Factory Patterns

Abstract Factory contains Factory Methods that create Command objects, which then dictate how users interact with an expression tree processing app.

These patterns enable extensibility of operations via new factory methods.