Factory Pattern
Mar. 8, 2007

What is it?

- returns an instance of one of several possible classes depending on the data provided to it
  - Usually all of the classes it returns have a common parent class and common methods, but each of them performs a task differently and is optimized for different kinds of data

A Closer Look

- x is a base class and classes xy and xz are derived from it.
- Factory is a class that decides which of these subclasses to return depending on the arguments you give it.
- On the right, we define a getClass method to be one that passes in some value abc, and that returns some instance of the class x.

More…

- Which one it returns doesn’t matter to the programmer since they all have “the same” methods, but different implementations.
- How it decides which one to return is entirely up to the factory.
  - It could be some very complex function but it is often quite simple.
An Example

- an entry form and we want to allow the user to enter name either
  - as “firstname lastname” or
  - as “lastname, firstname”
- decide the name order by whether there is a comma between the last and first name.

Let's look at some code

- start by defining a simple base class that takes a String and splits it (somehow) into two names:

```java
class Name {  
  // a simple class to take a string apart into two names  
  protected String last; // store last name here  
  protected String first; // store first name here  
  
  public String getFirst() {  
    return first; // return first name  
  }  
  public String getLast() {  
    return last; // return last name  
  }  
}  
```

- store the split first and last names in the Strings first and last, and, since the derived classes will need access to these variables, we'll make them protected.

A Derived Class “FirstFirst”

- In the FirstFirst class, we assume that everything before the last space is part of the first name:

```java
class FirstFirst extends Name {  
  // split first last  
  public FirstFirst(String s) {  
    int i = s.lastIndexOf(" "); // find space  
    if (i > 0) {  
      // left is first name  
      first = s.substring(0, i).trim(); // right is last name  
      last = s.substring(i+1).trim();  
    } else {  
      first = ""; // put all in last name  
      last = s; // if no space  
    }  
  }  
}  
```
Another Derived Class “LastFirst”

- LastFirst class, we assume that a comma delimits the last name.

```java
class LastFirst extends Namer {  //split last, first
    public LastFirst(String s) {  //find comma
        int i = s.indexOf(",");  //left is last name
        if (i > 0) {
            last = s.substring(0, i).trim();
            first = s.substring(i + 1).trim();
        } else {
            last = s;  // put all in last name
            first = "";  // if no comma
        }
    }
}
```

Lets Build the Factory!

- test for the existence of a comma and then return an instance of one class or the other.

```java
class NameFactory {  //returns an instance of LastFirst or FirstFirst
    public Namer getNamer(String entry) {  //comma determines name
        int i = entry.indexOf(“,“);  //comma determines name
        if (i > 0)
            return new LastFirst(entry);  //return one class
        else
            return new FirstFirst(entry);  //or the other
    }
}
```

Using the Factory

- initialize an instance of the factory class
  ```java
  NameFactory nFactory = new NameFactory();
  ```
- call the computeName method, which calls the getNamer factory method and then calls the first and last name methods of the class instance it returns
  ```java
  private void computeName() {
      //send the text to the factory and get a class back
      namer = nFactory.getNamer(entryField.getText());

      //compute the first and last names
      //using the returned class
      txFirstName.setText(namer.getFirstName());
      txLastName.setText(namer.getLastName());
  }
  ```

Fundamental Principle of Factory Patterns

- Create an abstraction which decides which of several possible classes to return, and – return one.
- Then you call the methods of that class instance without ever knowing which derived class you are actually using.
When to Use a Factory Pattern

- You should consider using a Factory pattern when
  - A class can’t anticipate which kind of class of objects it must create.
  - A class uses its subclasses to specify which objects it creates.
  - You want to localize the knowledge of which class gets created.