Creational Design Patterns

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

Creational Patterns

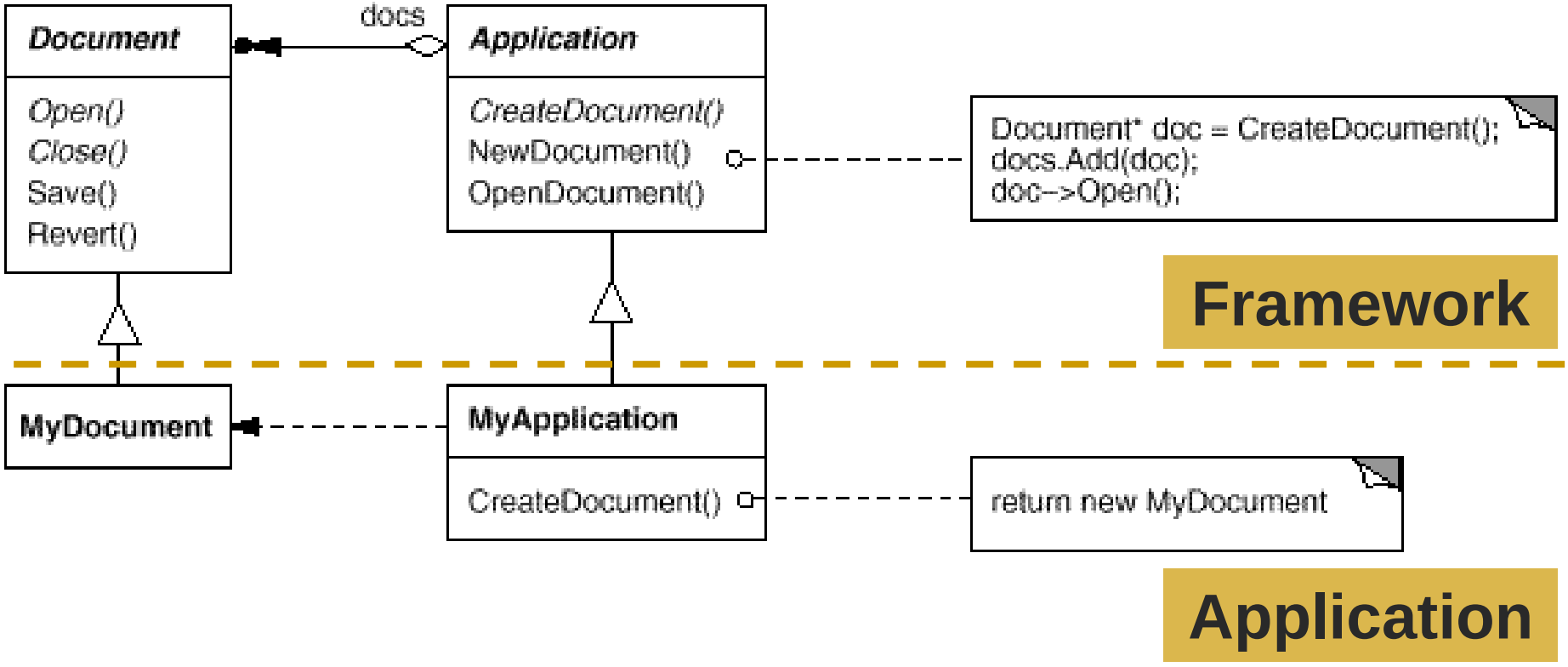
- Abstract Factory
- Builder
- **Factory Method**
- Prototype
- **Singleton**

Factory Method

Motivating Example

- Consider a framework that manages multiple documents
  - Applications create documents as required
- Factory Method encapsulates the knowledge of which document to create
  - The framework does not have this knowledge

Example of Application

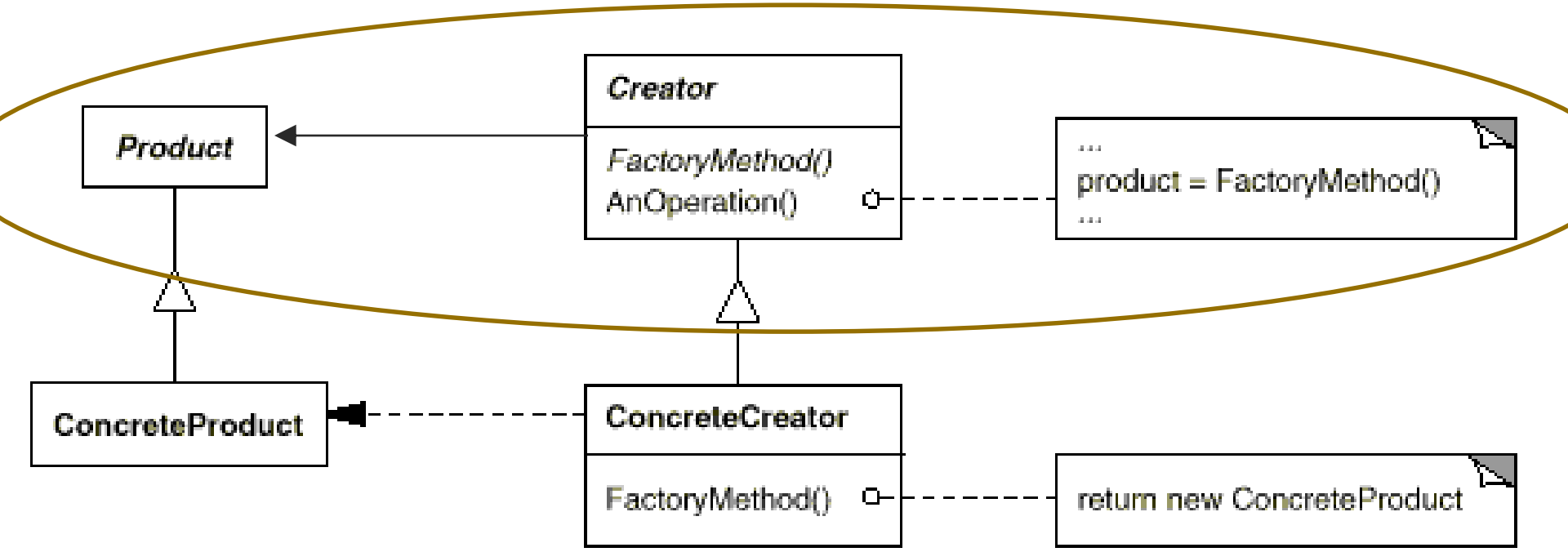


Documents can be of different kinds.

Factory Method

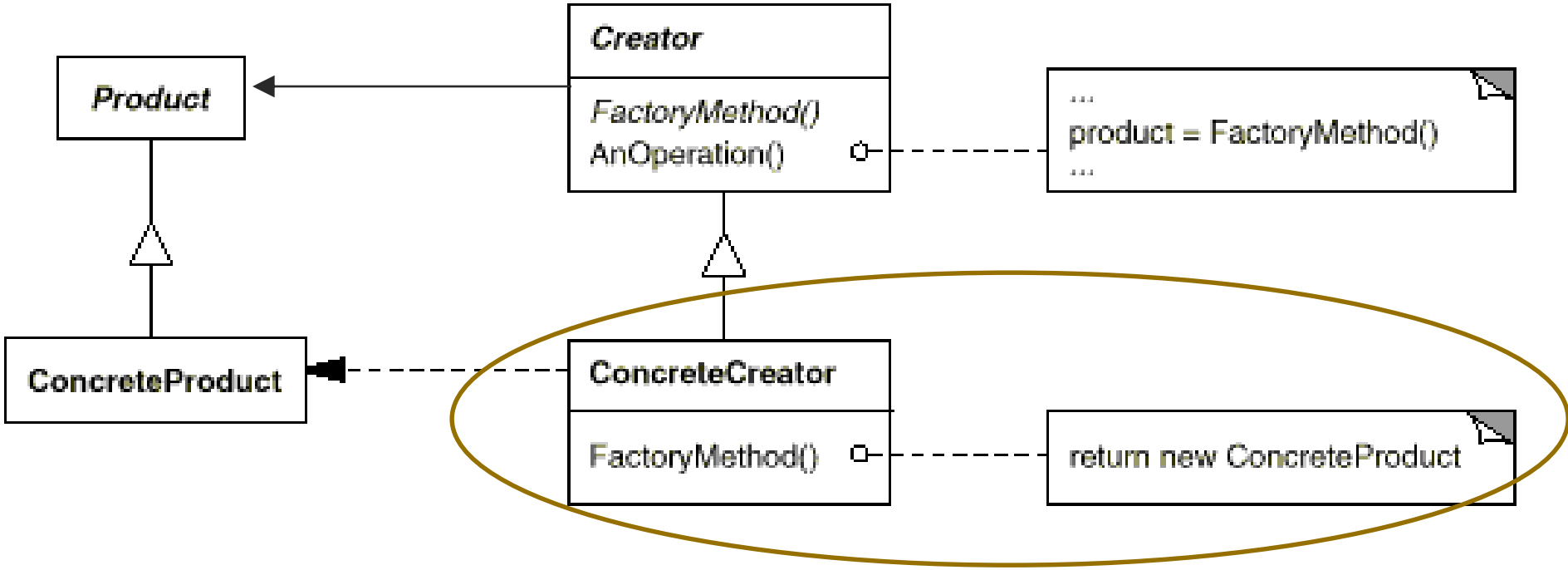
- Name
  - Factory Method
- Problem Description
  - Separate the creation of concrete objects from their abstract manipulation
- Solution (*next slide*)
- Consequences
  - Eliminate the need to bind application-specific classes into your framework code

Factory Method Solution



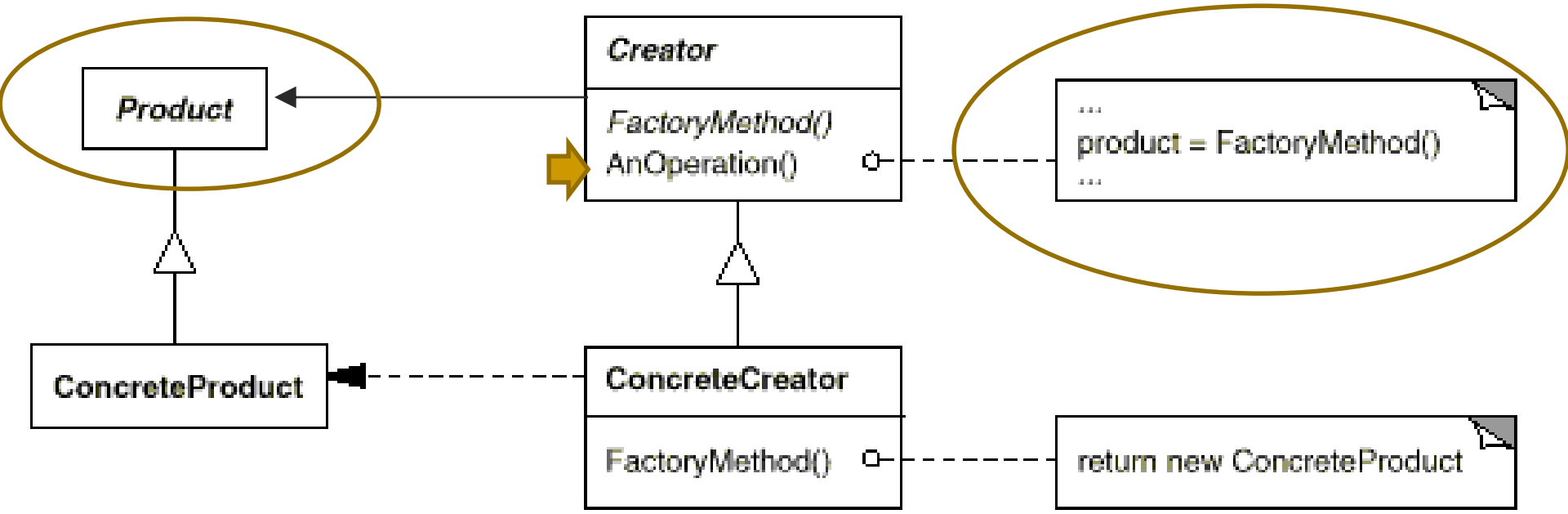
Super-classes do not know the specific product.

Concrete Factory Method



The factory method creates the product as required.

Abstract Product



The general product interface can be used, without knowing the specific product.

Singleton

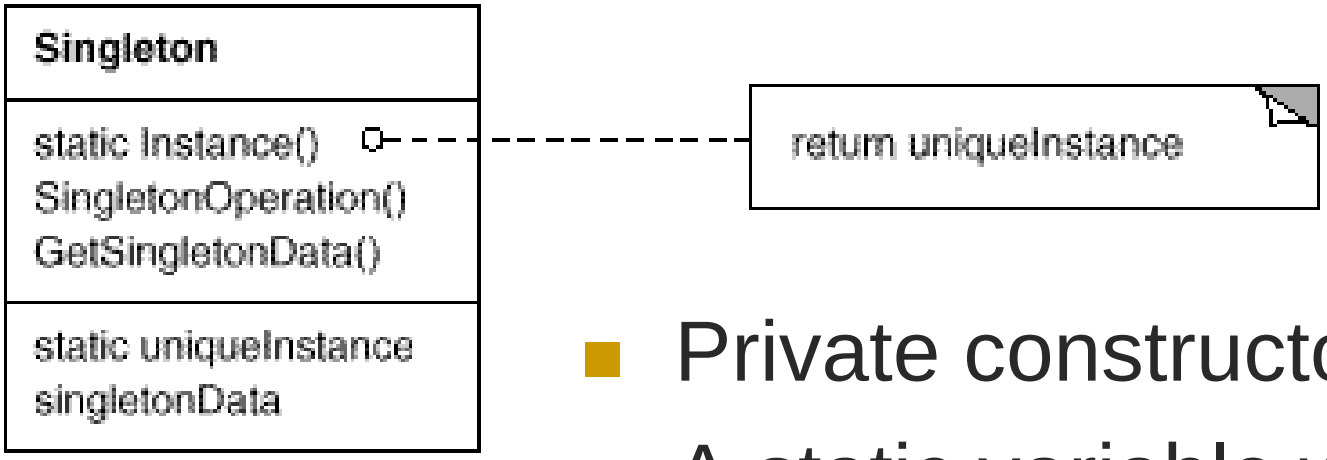
Singleton Context

- Some classes have exactly one instance
- Singleton makes the class itself responsible for keeping track of its sole instance

Singleton

- Name
  - Singleton
- Problem Description
  - Ensure that a class has only one instance
- Solution (*next slide*)
- Consequences
  - Controlled access to the sole instance
  - One access point to shared resources

Singleton Solution



- Private constructor
- A static variable with the solo instance
- The *instance()* method is static and public
  - It returns the solo instance

Example: ClasseSingleton

```
public class ClassSingleton {

  private static ClassSingleton instance;
  private int numInstances = 0;

  private ClassSingleton() {
    numInstances++;
  }

  public static ClassSingleton getInstance() {
    if (instance == null) instance = new ClassSingleton();
    return instance;
  }

  public void printNumInstances() {
    System.out.println("numInstances = "+numInstances);
  }
}
```

An static attribute is used to store the solo instance.

Example: ClasseSingleton

```
public class ClassSingleton {

   private static ClassSingleton instance;
   private int numInstances = 0;

   private ClassSingleton() {
      numInstances++;
   }

   public static ClassSingleton getInstance() {
      if (instance == null) instance = new ClassSingleton();
      return instance;
   }

   public void printNumInstances() {
      System.out.println("numInstances = "+numInstances);
   }
}
```

Constructor is private to avoid unexpected instantiations.

Exemplo: ClasseSingleton

```
public class ClassSingleton {

   private static ClassSingleton instance;
   private int numInstances = 0;

   private ClassSingleton() {
      numInstances++;
   }

   public static ClassSingleton getInstance() {
      if (instance == null) instance = new ClassSingleton();
      return instance;
   }

   public void printNumInstances() {
      System.out.println("numInstances = "+numInstances);
   }
}
```

The solo instance can only be accessed by the static *getInstance()* method.

Exemplo: ClasseSingletonTest

```
public class ClassSingletonTest {

   public static void main(String[] args) {
         ClassSingleton singleton = ClassSingleton.getInstance();
//       ClassSingleton singleton = new ClassSingleton();
         singleton.printNumInstances();
   }
}
```

If you try to directly instantiate a singleton class, a compilation error occurs (visibility).

Bibliography

- E. Gamma, R. Helm, R. Johnson, J. Vlissides. **Design Patterns**, 1st. Edition. Addison Wesley, 1994.
  - Factory Method and Singleton