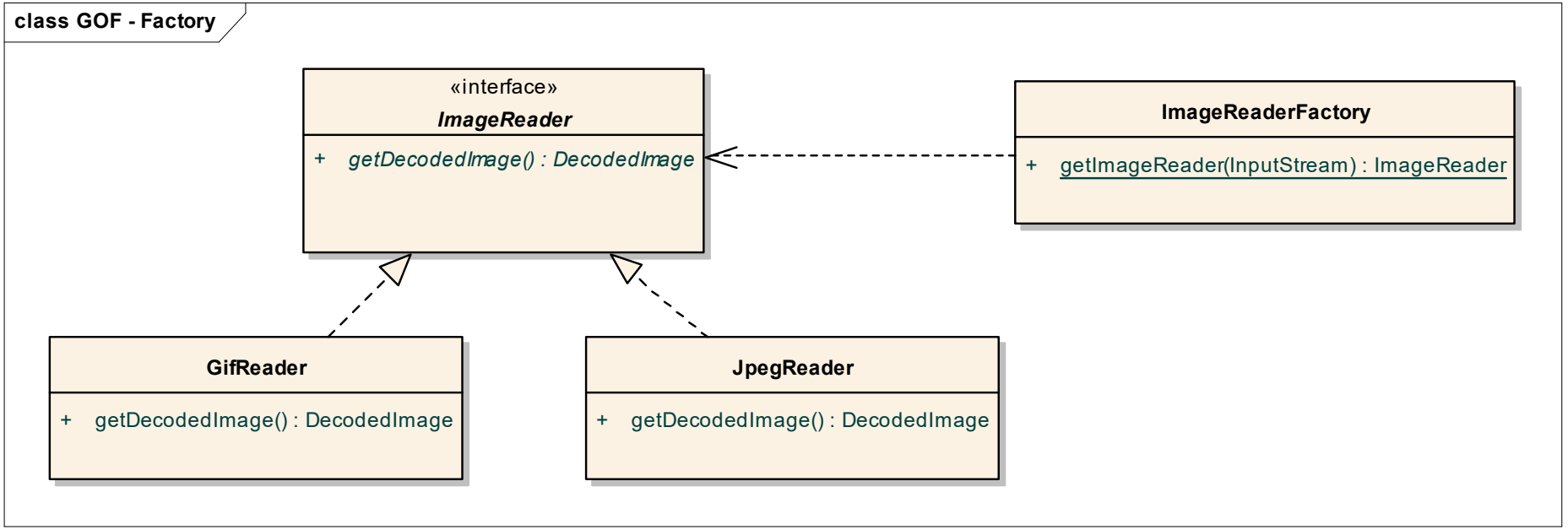
Padrões de Criação

- Factory
  - Fornece uma interface para criar um objeto, porém a decisão de qual classe será instanciada é decidida pelas subclasses
- Abstract Factory
  - Fornecem uma interface para criação de objetos sem especificar sua classe concreta
- Singleton
  - Garante que apenas uma classe possua uma única instância e oferece um acesso global à mesma
- Builder
  - Permite separar a construção de um objeto complexo de sua representação a fim de que diferentes objetos sejam criados através do mesmo processo
- Prototype
  - Permite que um objeto seja criado a partir de uma instância existente copiando suas propriedades

Factory

- Permite que um objeto seja criado sem que seja necessário informar a classe exata que será criada
- Separa a complexidade de criação do objeto
- Uma interface define um método padrão para criação
- Subclasses implentam este método e devolvem o objeto desejado
- A fábrica implementa o método criando o objeto conforme necessário



Factory
Exemplo

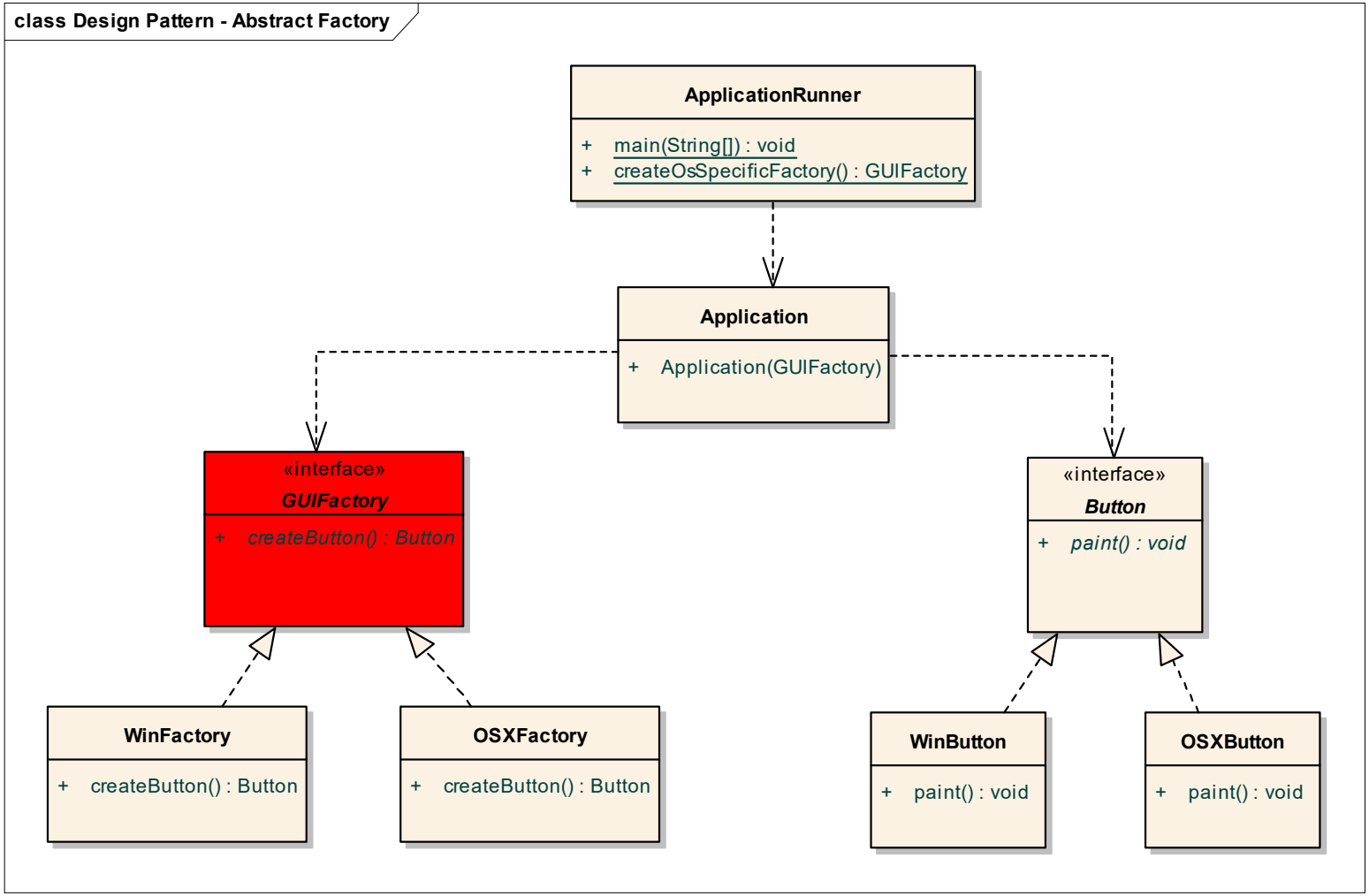
```
public interface ImageReader {
    public DecodedImage getDecodedImage();
}

public class GifReader implements ImageReader {
    public DecodedImage getDecodedImage()    {
        // ...
        return decodedImage;
    }
}

public class JpegReader implements ImageReader {
    public DecodedImage getDecodedImage()   {
        // ...
        return decodedImage;
    }
}
```

Factory
Exemplo

```
public class ImageReaderFactory {
    public static ImageReader getImageReader(InputStream is)     {
        int imageType = determineImageType(is);
        switch(imageType)         {
            case ImageReaderFactory.GIF:
                return new GifReader(is);
            case ImageReaderFactory.JPEG:
                return new JpegReader(is);
            // etc.
        }
    }
}
```

Abstract Factory

- Permite que objetos sejam criados de forma transparente caso exista um grupo de diferente fábricas



Abstract Factory

```
interface GUIFactory {
    public Button createButton();
}
class WinFactory implements GUIFactory {
    public Button createButton() {
        return new WinButton();
    }
}
class OSXFactory implements GUIFactory {
    public Button createButton() {
        return new OSXButton();
    }
}
```

Abstract Factory

```
interface Button {
    public void paint();
}
class WinButton implements Button {
    public void paint() {
        System.out.println("I'm a WinButton");
    }
}
class OSXButton implements Button {
    public void paint() {
        System.out.println("I'm an OSXButton");
    }
}
```

Abstract Factory

```
class Application {
    public Application(GUIFactory factory) {
        Button button = factory.createButton();
        button.paint();
    }
}

public class ApplicationRunner {
    public static void main(String[] args) {
        new Application(createOsSpecificFactory());
    }

    public static GUIFactory createOsSpecificFactory() {
        int sys = readFromConfigFile("OS_TYPE");
        if (sys == 0) {
            return new WinFactory();
        } else {
            return new OSXFactory();
        }
    }
}
```

Singleton

- Objetivo
  - Garante que existirá uma única instância de um objeto de uma classe e permite um acesso global ao mesmo
- Motivação
  - Em muitas situações é necessário um único objeto. Exemplos:
    - O objeto que representa um sistema de arquivos do Sistema Operacional
    - Um objeto que representa um arquivo de configuração de uma aplicação
    - Um objeto que representa uma conexão com um banco de dados

Singleton
Exemplo

```
public class Singleton {
  //instância de um objeto da classe Singleton
  //inicializada com a chamada do construtor
  private static Singleton instance = new Singleton();

  //Construtor privado impede criação de objetos desta classe
  private Singleton() {
    //lógica para criação do objeto
  }

  //método estático que retorna a única instância da classe
  public static Singleton getInstance() {
    return instance;
  }
}
```

Singleton
Exemplo - Outra abordagem

```
public class Singleton {
  // Private constructor prevents instantiation from other classes
  private Singleton() {
  }

  /**
   * SingletonHolder is loaded on the first execution of Singleton.getInstance()
   * or the first access to SingletonHolder.INSTANCE, not before.
   */
  private static class SingletonHolder {
    public static final Singleton instance = new Singleton();
  }

  public static Singleton getInstance() {
    return SingletonHolder.instance;
  }
```