1) Este padrão define uma interface para criar uma família de objetos relacionados ou dependentes e que não especificam suas classes concretas, em resumo, uma interface para cada tipo de criação objeto básico.

2) Quando queremos facilitar o câmbio das famílias de produtos e no ato de um sistema independente na criação de seus produtos e também quando um certo sistema tem que se configurar com uma família entre várias famílias de produtos, e quando queremos fornecer uma biblioteca de classes e queremos apenas revelar sua interface e não sua implementação.

3) Exemplo de implementação: criação de labirintos:

```
public interface FactoryDeLabirintoIF {
  public LabirintoIF criaLabirinto();
  public SalaIF criaSala(int númeroDaSala);
  public ParedeIF criaParede() {
  public PortaIF criaPorta(SalaIF sala1, SalaIF sala2) {
}

public class FactoryDeLabirinto implements FactoryDeLabirintoIF {
  private static FactoryDeLabirintoIF instânciaÚnica = null;

  private FactoryDeLabirinto() {}

  public static FactoryDeLabirintoIF getInstance(String tipo) {
    if(instânciaÚnica == null) {
      if(tipo.equals("perigoso")) {
        instânciaÚnica = new FactoryDeLabirintoPerigoso();
      } else if(tipo.equals("encantado")) {
        instânciaÚnica = new FactoryDeLabirintoEncantado();
      } else {
        instânciaÚnica = new FactoryDeLabirinto();
      }
    }
    return instânciaÚnica;
  }

  // Factory Methods
  // Tem default para as Factory Methods
  public LabirintoIF criaLabirinto() {
    return new Labirinto();
  }

  public SalaIF criaSala(int númeroDaSala) {
    return new Sala(númeroDaSala);
  }

  public ParedeIF criaParede() {
    return new Parede();
  }

  public PortaIF criaPorta(SalaIF sala1, SalaIF sala2) {
    return new Porta(sala1, sala2);
  }
}
```

- A nova versão de montaLabirinto recebe um FactoryDeLabirintoIF como parâmetro e cria um labirinto

```
public class Jogo implements JogoIF {
  // Observe que essa função não tem new: ela usa uma Abstract Factory
```

```
  // Esta é a *única* diferença com relação à versão original
  // Observe como montaLabirinto acessa a factory (através de um singleton)
  public LabirintoIF montaLabirinto(FactoryDeLabirintoIF factory) {
    LabirintoIF umLabirinto = factory.criaLabirinto();
    SalaIF sala1 = factory.criaSala(1);
    SalaIF sala2 = factory.criaSala(2);
    PortaIF aPorta = factory.criaPorta(sala1, sala2);

    umLabirinto.adicionaSala(sala1);
    umLabirinto.adicionaSala(sala2);

    sala1.setVizinho(NORTE, factory.criaParede());
    sala1.setVizinho(LESTE, aPorta);
    sala1.setVizinho(SUL, factory.criaParede());
    sala1.setVizinho(OESTE, factory.criaParede());

    sala2.setVizinho(NORTE, factory.criaParede());
    sala2.setVizinho(LESTE, factory.criaParede());
    sala2.setVizinho(SUL, factory.criaParede());
    sala2.setVizinho(OESTE, aPorta);

    return umLabirinto;
  }
}
```

- Para criar um labirinto encantado, criamos uma factory concreta como subclasse de FactoryDeLabirinto

```
public class FactoryDeLabirintoEncantado extends FactoryDeLabirinto {
  public SalaIF criaSala(int númeroDaSala) {
    return new salaEncantada(númeroDaSala, jogaEncantamento());
  }
  public PortaIF criaPorta(SalaIF sala1, SalaIF sala2) {
    return new portaPrecisandoDeEncantamento(sala1, sala2);
  }
  protected EncantamentoIF jogaEncantamento() {
    ...
  }
}
```

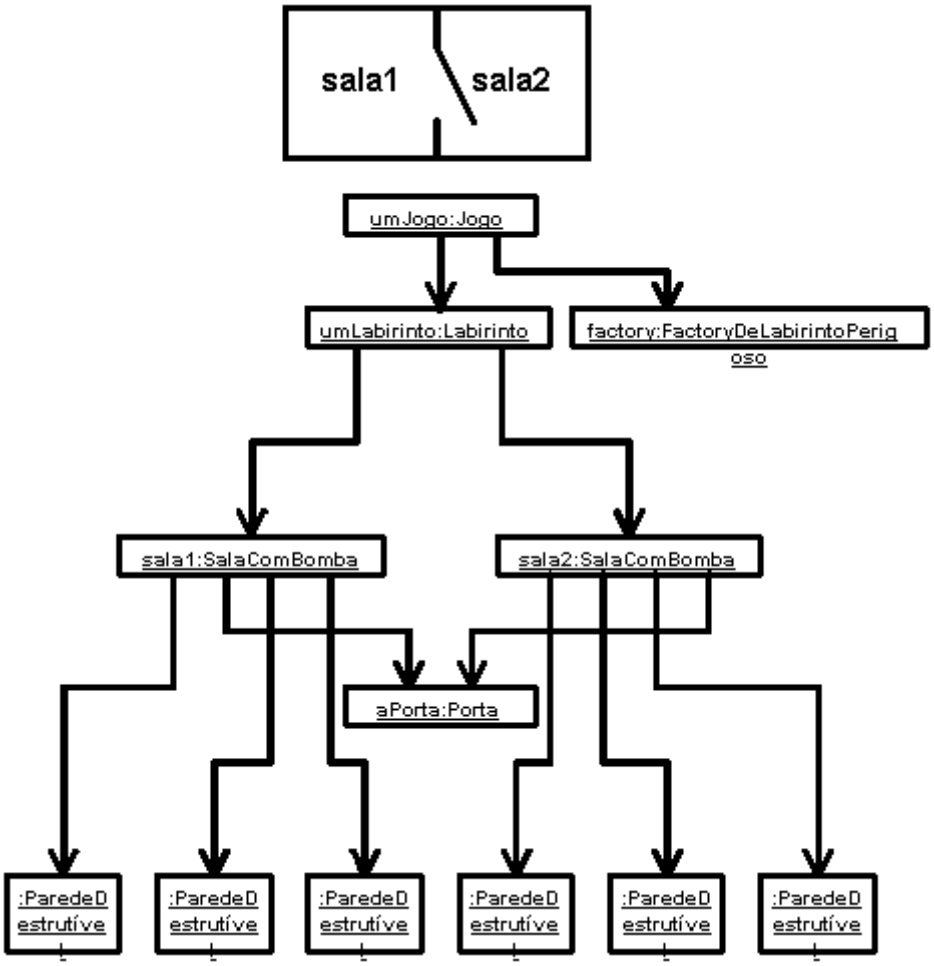
- Para criar um labirinto perigoso, criamos uma *outra* factory concreta como subclasse de FactoryDeLabirinto

```
public class FactoryDeLabirintoPerigoso extends FactoryDeLabirinto {
  public ParedeIF criaParede() {
    return new paredeDestruível();
  }
  public SalaIF criaSala(int númeroDaSala) {
    return new salaComBomba(númeroDaSala);
  }
}
```

- Finalmente, podemos jogar:

```
  JogoIF umJogo = new Jogo();
  FactoryDeLabirinto factory = FactoryDeLabirinto.getInstance("perigoso");
  jogo.montaLabirinto(factory);
```

- Poderíamos jogar um jogo encantado com uma mudança muito simples ao código acima
- O diagrama de objetos tem um objeto a mais comparado com a versão original



Terminologia (UML):