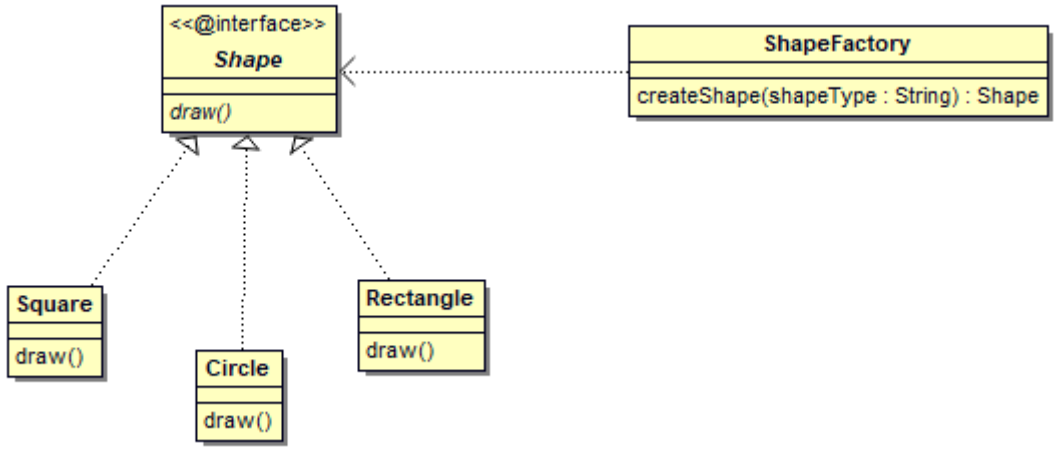
Pattern Factory

Supposons un logiciel de dessin où on demande au développeur de pouvoir dessiner des rectangles. L'idée de base consiste à créer une classe Rectangle et à l'utiliser avec l'opérateur new.
Que se passe-t-il si on veut faire évoluer l'application pour dessiner des cercles : évidemment on peut créer une classe Cercle et créer des objets avec new, et de même si on veut des objets de type Carre…
Le programme « client » créera directement les objets en utilisant l'opérateur new et implicitement les divers constructeurs.

Le patron de conception **Factory** propose de remplacer ces appels aux constructeurs des diverses classes par une méthode spéciale de « fabrique » pour créer les divers objets. Cette méthode utilise toujours l'opérateur new et elle retourne les objets voulus. Cette méthode de « fabrique » sera placée dans une classe spécifique « factory ».



Le code Java

```
package factory;

public interface Shape {
	void draw();
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
package factory;

public class Rectangle implements Shape {

	@Override
	public void draw() {
		// TODO Auto-generated method stub
		System.out.println("Je dessine le RECTANGLE");
	}
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
package factory;
```

```java
public class Square implements Shape {

	@Override
	public void draw() {
		// TODO Auto-generated method stub
		System.out.println("Je dessine le CARRE");
	}
}
```

//

```java
package factory;

public class Circle implements Shape {

	@Override
	public void draw() {
		// TODO Auto-generated method stub
		System.out.println("Je dessine le CERCLE");
	}
}
```

//

```java
package factory;

public class ShapeFactory {
	/*
	 * Cette méthode reçoit en paramètre le nom de la forme à dessiner Elle retourne
	 * un objet "forme" correspondant au paramètre donné, null si le paramètre donné
	 * ne correspond pas à une des 3 formes
	 */
	public Shape createShape(String shapeType) {

		if (shapeType.equalsIgnoreCase("CIRCLE")) {
			return new Circle();
		}
		if (shapeType.equalsIgnoreCase("RECTANGLE")) {
			return new Rectangle();
		}
		if (shapeType.equalsIgnoreCase("SQUARE")) {
			return new Square();
		}
		return null;
	}
}
```

////////////////////////// Exemple de main //

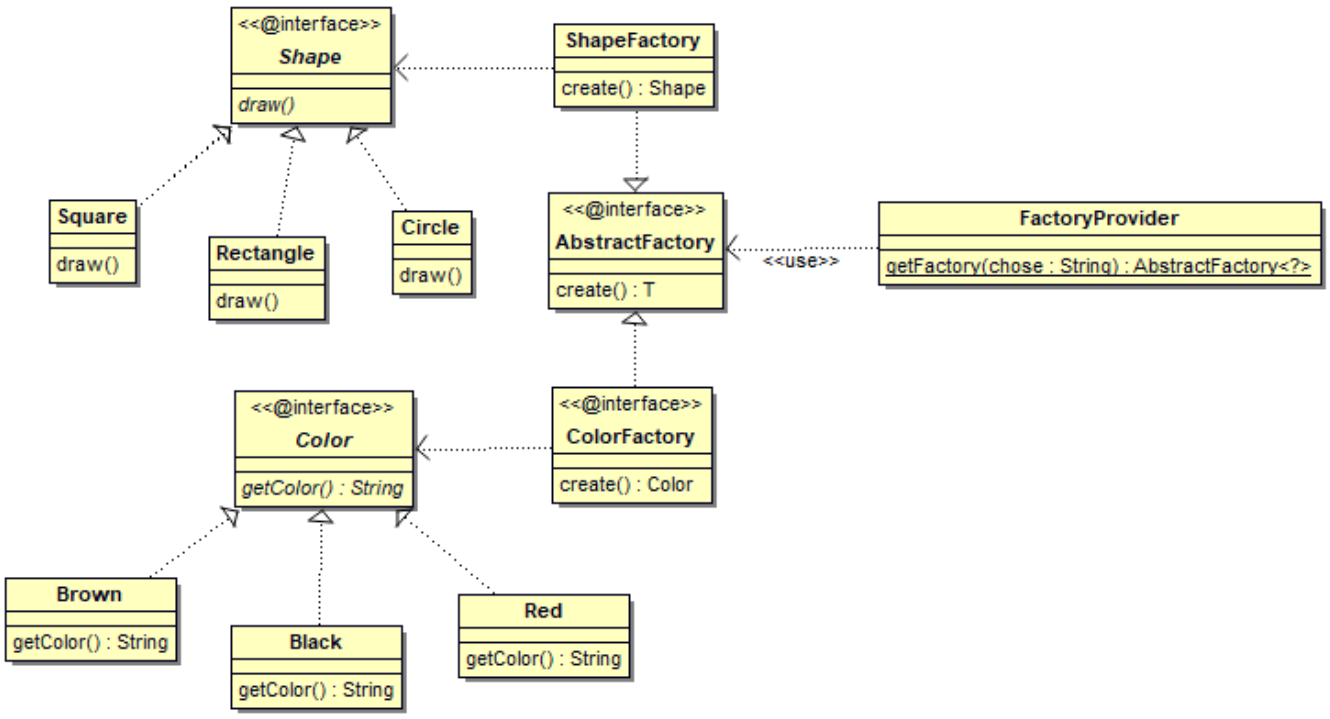
```java
package factory;

public class MainPatternFactoryDemo {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
```

```
        ShapeFactory shapeFactory = new ShapeFactory();

        // get an object of Circle and call its draw method.
        Shape shape1 = shapeFactory.createShape("CIRCLE");
        // call draw method of Circle
        shape1.draw();

        // get an object of Rectangle and call its draw method.
        Shape shape2 = shapeFactory.createShape("RECTANGLE");
        // call draw method of Rectangle
        shape2.draw();

        // get an object of Square and call its draw method.
        Shape shape3 = shapeFactory.createShape("SQUARE");
        // call draw method of square
        shape3.draw();
    }
}
```

La classe MainPatternFactoryDemo joue le rôle de programme client. Le but du pattern factory est de ne pas exposer la logique de création des divers objets au programme client : il doit passer par la factory pour créer les objets.

Le pattern Abstract Factory

Ce pattern s'applique dès lors qu'il faut créer des objets de « familles » différentes, par exemple ici des objets de type Shape et Color.
La méthode **getFactory(String chose)** du **FactoryProvider** renvoie le « bon » Factory, soit ColorFactory, soit ShapeFactory en fonction de l'argument **chose** passé.



Le code java

L'interface Shape et les 3 classes Square, Rectangle et Circle sont inchangées.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
package abstractfactory;

public class FactoryProvider {
	public static AbstractFactory<?> getFactory(String choice) {

		if ("Shape".equalsIgnoreCase(choice)) {
			return new ShapeFactory();
		} else if ("Color".equalsIgnoreCase(choice)) {
			return new ColorFactory();
		}
		return null;
	}
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
package abstractfactory;

public class ShapeFactory implements AbstractFactory{
	public Shape create(String shapeType) {

		if (shapeType.equalsIgnoreCase("CIRCLE")) {
			return new Circle();
		}
		if (shapeType.equalsIgnoreCase("RECTANGLE")) {
			return new Rectangle();
		}
		if (shapeType.equalsIgnoreCase("SQUARE")) {
			return new Square();
		}
		return null;
	}
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
package abstractfactory;

public class ColorFactory implements AbstractFactory<Color> {

	@Override
	public Color create(String chose) {
		if (chose.equalsIgnoreCase("BLACK"))
			return new Black();
		if (chose.equalsIgnoreCase("RED"))
			return new Red();
		return null ;
	}
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
package abstractfactory;

public interface Color {
	String getColor() ;
}
```
//
```
package abstractfactory;

public class Black implements Color {

	@Override
	public String getColor() {
		// TODO Auto-generated method stub
		return "BLACK";
	}
}
```
//
```
package abstractfactory;

public class Red implements Color {

	@Override
	public String getColor() {
		// TODO Auto-generated method stub
		return "RED";
	}
}
```
//

La classe cliente, le main

```
package abstractfactory;

public class MainPatternAbstractFactory {

   public static void main(String[] args) {
		// TODO Auto-generated method stub

	ShapeFactory shapeFactory =(ShapeFactory) FactoryProvider.getFactory("shape");
	Square s = (Square) shapeFactory.create("square");
	s.draw();
	Circle c = (Circle) shapeFactory.create("circle");
	c.draw();

	ColorFactory colorFactory = (ColorFactory)FactoryProvider.getFactory("color");
	Black noir = (Black) colorFactory.create("black");
	System.out.println(noir.getColor());
	Red rouge = (Red) colorFactory.create("red");
	System.out.println(rouge.getColor());
   }
}
```