PATRÓN SIMPLE FACTORY

MOTIVACIÓN

En determinadas situaciones, nos interesa crear objetos de distintas clases en función de algún dato o bajo determinadas condiciones. ¿Cómo podemos simplificar el código?

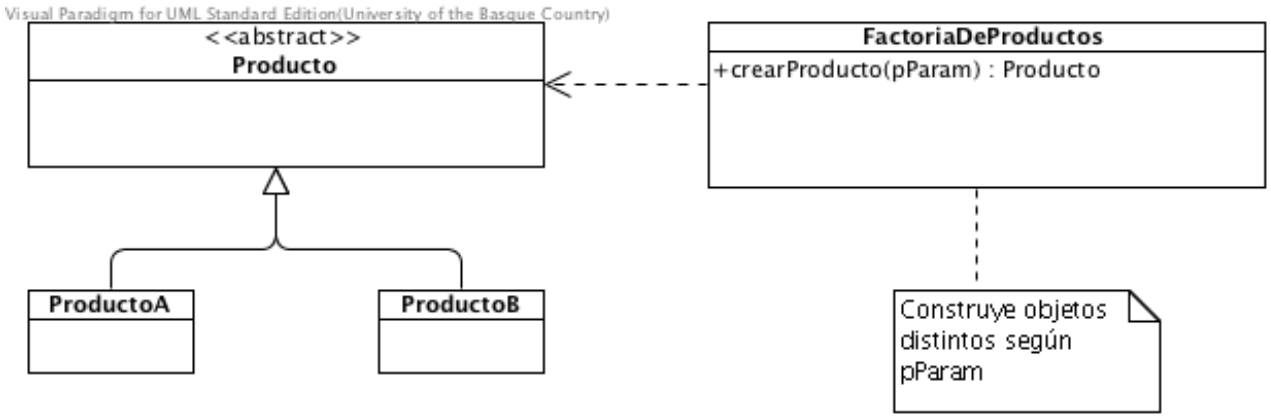
Vamos a plantearnos la siguiente situación. El juego *Arcanoid* consiste en destruir una pared de ladrillos (si no son irrompibles) haciendo que una pelota rebote en cada uno de los ladrillos. Los ladrillos pueden ser de diversos tipos. ¿Cómo podemos construir la pared inicial de una manera sencilla?

OBJETIVOS

- Encapsular el conocimiento sobre el tipo de objeto a crear
- Controlar la creación de objetos (seguridad, estadísticas, etc.)
- Usar distintos constructores sin preocuparnos de la clase

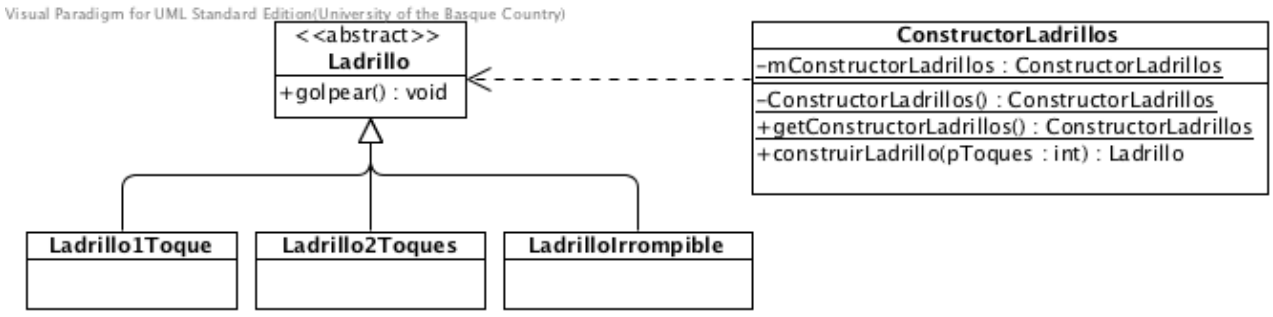
DESCRIPCIÓN

El patrón *Simple Factory* es uno de los denominados *patrones de creación*, que hacen referencia a la forma en la que se crean objetos. Esta clase permite crear objetos de otras clases.



EJEMPLOS

En la implementación del *Arcanoid*:



APLICACIÓN 1

```
package packArcanoid;

public class ConstructorLadrillos {
	private static ConstructorLadrillos mConstructorLadrillos =
					new ConstructorLadrillos();

	private ConstructorLadrillos() {
	}

	public static ConstructorLadrillos getConstructorLadrillos() {
		return mConstructorLadrillos;
	}

	public Ladrillo construirLadrillos(int pTipo) {
		switch (pTipo) {
		case 0:
			return new LadrilloIrrompible();
		case 1:
			return new Ladrillo1Toque();
		default:
			// pTipo=2
			return new Ladrillo2Toques();
		}
	}
}
```

```
package packArcanoid;

import java.util.Random;

public class ProgramaPrincipal {
	public static void main(String[] args) {
		Random random = new Random();
		ConstructorLadrillos cons =
				ConstructorLadrillos.getConstructorLadrillos();
		Ladrillo matriz[][] = new Ladrillo[3][3];
		int numToques;
		for (int i = 0; i < matriz.length; i++) {
			for (int j = 0; j < matriz[i].length; j++) {
				numToques = random.nextInt(3);
				matriz[i][j] = cons.construirLadrillos(numToques);
			}
		}
	}
}
```

APLICACIÓN 2

Utilizando *Java Reflection*[1] se puede conseguir un constructor de ladrillos que no haya que modificar cuando cambie la jerarquía de ladrillos.

```
package packArcanoid;

import java.lang.reflect.Constructor;
```

[1] http://java.sun.com/developer/technicalArticles/ALT/Reflection/

```
public class ConstructorLadrillosV2 {
	private static ConstructorLadrillosV2 mConstructorLadrillos =
				new ConstructorLadrillosV2();

	private ConstructorLadrillosV2() {
	}

	public static ConstructorLadrillosV2 getConstructorLadrillos() {
		return mConstructorLadrillos;
	}

	public Ladrillo construirLadrillos(String pTipo) {
		try {
			Class ladClass = Class.forName(pTipo);
			Class[] intArgClass = new Class[] {};
			Object[] intArgs = new Object[] {};
			Constructor intAConstructor;
			intAConstructor = ladClass.getConstructor(intArgClass);
			Ladrillo ladrillo = (Ladrillo)
				intAConstructor.newInstance(intArgs);
			return ladrillo;
		} catch (Exception e) {
			throw new IllegalArgumentException();
		}
	}
}


package packArcanoid;
import java.util.Random;

public class ProgramaPrincipal2 {


	public static void main(String[] args) {
		String tipoLadrillos[] = new String[]
				{"packArcanoid.LadrilloIrrompible",
				 "packArcanoid.Ladrillo1Toque",
				 "packArcanoid.Ladrillo2Toques"};
		Random random = new Random();
		ConstructorLadrillosV2 cons =
			ConstructorLadrillosV2.getConstructorLadrillos();
		Ladrillo matriz[][] = new Ladrillo[3][3];
		int numToques;
		for (int i = 0; i < matriz.length; i++) {
			for (int j = 0; j < matriz[i].length; j++) {
				numToques = random.nextInt(3);
				matriz[i][j] =
			cons.construirLadrillos(tipoLadrillos[numToques]);
			}
		}
	}
}
```