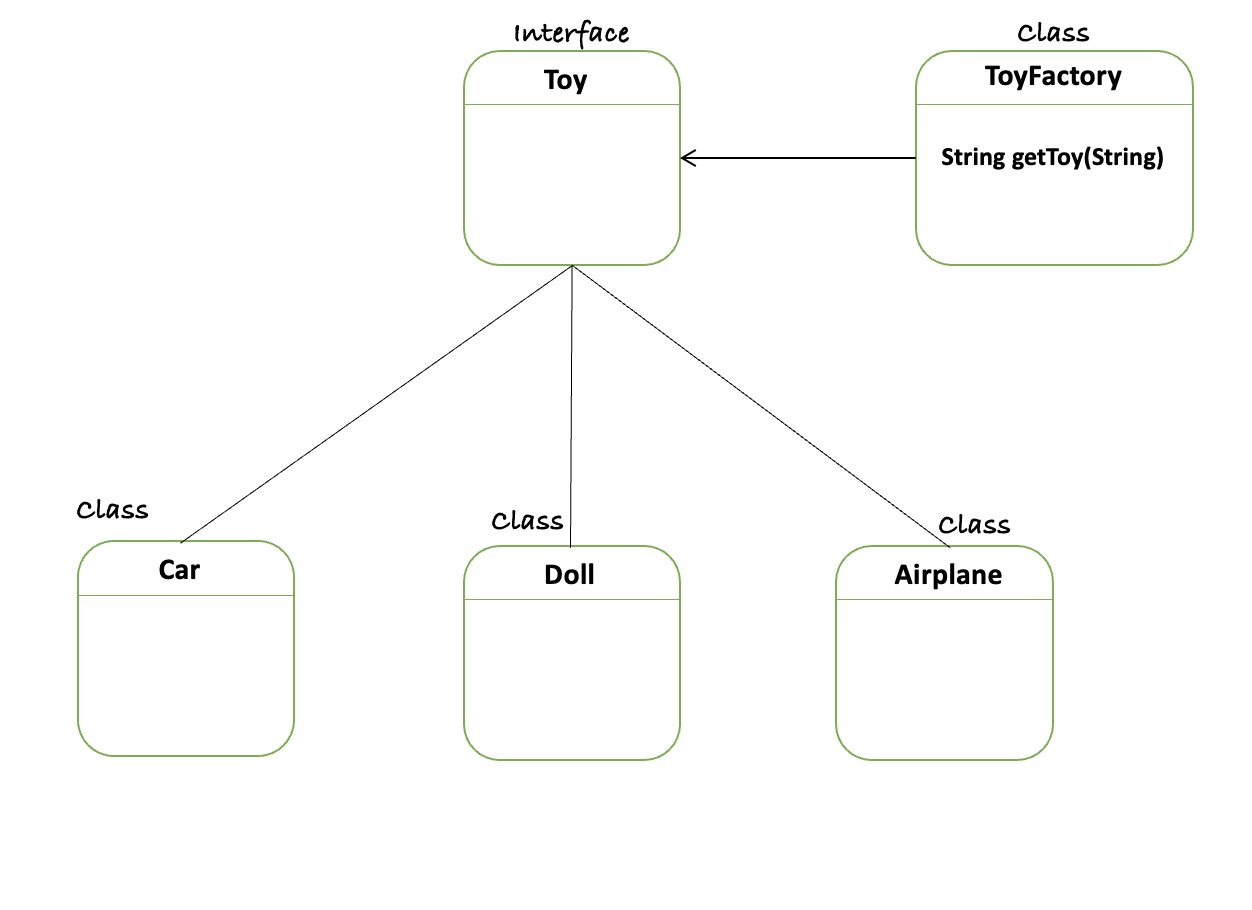
**Factory Design Pattern Concept (Layman View)**

Imagine you run a toy shop, and you want to make different types of toys like cars, dolls, and airplanes. Each toy type requires different materials and a different process to make. Now, instead of having one big machine that tries to make all types of toys, which can get quite complicated, you decide to have separate "factories" for each type of toy. Each factory specializes in making one type of toy and knows exactly what materials and steps are needed to make that toy.

In software terms, the factory method is a design pattern. A design pattern is like a blueprint for solving common problems in software design. The factory method is used when we have a superclass (a general category) with multiple subclasses (specific categories). Instead of asking the superclass to create an object, we delegate the creation to subclasses.

Back to our toy shop analogy, the "superclass" would be the general concept of a toy, and the "subclasses" would be specific types of toys like cars, dolls, and airplanes. The factory method allows each type of toy to have its own factory that knows how to create that specific toy.

In real-world software, this means that when we need to create an object (like a toy in our analogy), we use a method (the factory) in the subclass to do it. This makes our code more modular (organized into separate parts), easier to manage, and adaptable. If we need to add a new type of toy to our shop, we just add a new factory without messing with the existing ones.



// Toy.java

public interface Toy {

String createToy();

}

// Car.java

public class Car implements Toy {

@Override

public String createToy() {

return "Car toy created with wheels and a body";

}

}

// Doll.java

public class Doll implements Toy {

@Override

public String createToy() {

return "Doll toy created with fabric and yarn";

}

}

// Airplane.java

public class Airplane implements Toy {

@Override

public String createToy() {

return "Airplane toy created with wings and a fuselage";

}

}

// ToyFactory.java

public class ToyFactory {

public Toy getToy(String toyType) {

switch (toyType) {

case "car":

return new Car();

case "doll":

return new Doll();

case "airplane":

return new Airplane();

default:

throw new IllegalArgumentException("Toy type not supported");

}

}

}

// Main.java

public class Main {

public static void main(String[] args) {

ToyFactory factory = new ToyFactory();

Toy car = factory.getToy("car");

System.out.println(car.createToy());

Toy doll = factory.getToy("doll");

System.out.println(doll.createToy());

Toy airplane = factory.getToy("airplane");

System.out.println(airplane.createToy());

}

}