Certainly! In TypeScript, decorators are classified into four main types based on what they can decorate. These types are:

1. **Class Decorators**: Applied to classes and class constructors. They receive the constructor function of the class as their target.

2. **Method Decorators**: Applied to methods within a class. They receive three parameters: the target object (the prototype of the class or constructor function if the method is static), the name of the method, and a property descriptor for the method.

3. **Property Decorators**: Applied to properties of a class. They receive two parameters: the target object (the prototype of the class) and the name of the property.

4. **Parameter Decorators**: Applied to parameters of a method or constructor within a class. They receive three parameters: the target object, the name of the method or constructor, and the index of the parameter.

Here's a brief overview of each type:

1. Class Decorators:
```typescript
function ClassDecorator(target: Function) {
    // Decorator logic for classes
}

@ClassDecorator
class MyClass {
    // Class definition
}
```

2. Method Decorators:
```typescript
function MethodDecorator(target: any, methodName: string, descriptor: PropertyDescriptor) {
    // Decorator logic for methods
}

class MyClass {
    @MethodDecorator
    myMethod() {
        // Method definition
    }
}
```

3. Property Decorators:
```typescript
function PropertyDecorator(target: any, propertyName: string) {
    // Decorator logic for properties
}

class MyClass {
    @PropertyDecorator
    myProperty: string;
}
```

4. Parameter Decorators:
```typescript
function ParameterDecorator(target: any, methodName: string, parameterIndex: number) {
    // Decorator logic for parameters
}

class MyClass {
    myMethod(@ParameterDecorator param1: string, @ParameterDecorator param2: number) {
        // Method definition
    }
}
```

Certainly! Here's one more example for each type of decorator:

1. Class Decorator:
Class decorators can be used for various purposes such as logging, dependency injection, or applying metadata to classes. Here's an example of a simple logging class decorator:

```typescript
function LogClass(target: Function) {
    console.log(`Class ${target.name} is being instantiated.`);
}

@LogClass
class MyClass {
    // Class definition
}

// Output: Class MyClass is being instantiated.
```

2. Method Decorator:
Method decorators can be used to log method calls, validate inputs, or enforce access control. Here's an example of a method decorator that logs method calls:

```typescript
function LogMethod(target: any, methodName: string, descriptor: PropertyDescriptor) {
    const originalMethod = descriptor.value;

    descriptor.value = function(...args: any[]) {
        console.log(`Calling method ${methodName} with arguments:`, args);
        const result = originalMethod.apply(this, args);
        console.log(`Method ${methodName} returned:`, result);
        return result;
    };

    return descriptor;
}

class Calculator {
    @LogMethod
    add(a: number, b: number): number {
        return a + b;
    }
}

const calculator = new Calculator();
calculator.add(3, 5);

// Output:
// Calling method add with arguments: [3, 5]
// Method add returned: 8
```

3. Property Decorator:
Property decorators can be used for validation, data transformation, or implementing computed properties. Here's an example of a property decorator that converts a property value to uppercase:

```typescript
function Uppercase(target: any, propertyName: string) {
    let value: string = target[propertyName];

    const getter = function() {
```

```
        return value;
    };

    const setter = function(newValue: string) {
        value = newValue.toUpperCase();
    };

    Object.defineProperty(target, propertyName, {
        get: getter,
        set: setter,
        enumerable: true,
        configurable: true
    });
}

class Example {
    @Uppercase
    name: string = "john";
}

const example = new Example();
console.log(example.name); // Output: JOHN
example.name = "Alice";
console.log(example.name); // Output: ALICE
```

4. Parameter Decorator:
Parameter decorators can be used for validation, logging, or implementing aspect-oriented features. Here's an example of a parameter decorator that logs the value of a method parameter:

```typescript
function LogParameter(target: any, methodName: string, parameterIndex: number) {
    console.log(`Parameter ${parameterIndex} of method ${methodName} is being accessed.`);
}

class Example {
    greet(@LogParameter message: string): void {
        console.log(message);
    }
}

const example = new Example();
example.greet("Hello, world!");
```

```
// Output:
// Parameter 0 of method greet is being accessed.
// Hello, world!
```

These examples demonstrate how decorators can be applied to different parts of your TypeScript codebase to modify behavior, enforce rules, or add additional functionality.