Generics

Interfaces, Generic Functions and Classes

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

http://softuni.bg

Have a Question?

Software University

sli.do

#TypeScript

Table of Content

Design Patterns

Common OOP Design Patterns

- **Singleton Pattern**: ensures a class has only one instance and provides a global point of access
- **Factory Method Pattern**: defines an interface for creating an object but allows subclasses to alter the type of objects that will be created

Common OOP Design Patterns

Software University

- **Observer Pattern**: defines a dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- **Strategy Pattern**: defines a family of algorithms, encapsulates each one, and makes them interchangeable

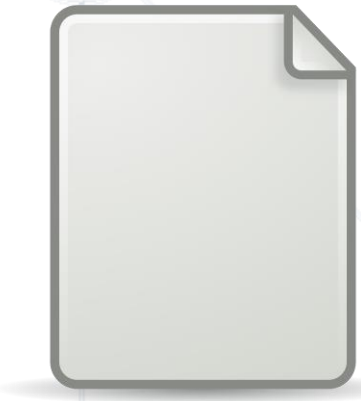
Common OOP Design Patterns

- **Decorator Pattern**: attaches new functionalities to an object dynamically without modifying its structure
- **Adapter Pattern**: allows incompatible interfaces to work together by providing a wrapper around the incompatible object

Generics

Definition

- Used to build **reusable** software components
- The components will work with **multitude** of type instead of a single type
- Defined by type variable - **<LETTER>**
- Follow the **DRY** (**D**on't **R**epeat **Y**ourself) principle
- Allow us to **abstract** the type
- Generics can be applied to **functions**, **classes** and **interfaces**



Example: Generic vs Non-Generic

- Generic

```
function echo<T>(arg: T): T {
    console.log(typeof arg);
    // It will print number and
string when the function is
invoked
    return arg;
}
echo(11111);
echo('Hello');
```

- Non-generic

```
function echo(arg: number): number {
    return arg;
}
```

```
function echo(arg: string): string {
    return arg;
}
```

Generic Functions

Software University

- Generic functions allow us to work with user input with **unknown** data type
- It is a way of telling the function that whatever **type** is **passed** to it the **same** type shall be **returned**
- Put some **constraints** to user input
- We can put **more than one** type variable in the generic function

Example: Generic Functions

```
const takeLast = <T>(array: T[]) => {
    return array.pop();
}
const sample = takeLast(['Hello', 'World', 'TypeScript']);
const secondSample = takeLast([1, 2, 3, 4]);
console.log(sample, secondSample); //TypeScript, 4
```

```
const makeTuple = <T, V>(a: T, b: V) => {
    return [a, b];
}
const firstTuple = makeTuple(1, 2);
const secondTuple = makeTuple('a', 'b');
console.log(firstTuple, secondTuple); //[1, 2], [a, b]
```

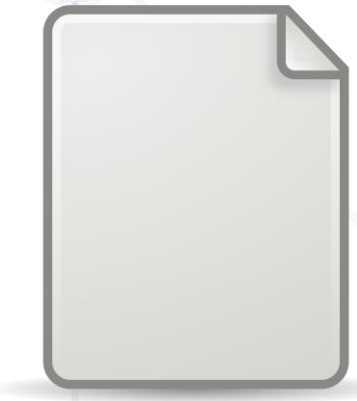
Generic Interfaces

- Using **generic interfaces** we can define **generic functions** too

```
interface GenericConstructor<T, V> {
    (arg: T, param: V): [T, V];
}

const generatedFn: GenericConstructor<string, string> = <T, V>(arg: T, param: V)
 => {
       return [arg, param];
}

const sample = generatedFn('Hello', 'World');
console.log(sample); // [Hello, World]
```

Generic Classes

- Generics can be used on:
  - The **properties** of the class
  - The **methods** of the class
- To define generic class we put **<LETTER> after the name of the class**
- We can use **multiple** type variables
- Generic classes can implement **generic interfaces**



Example: Generic Class Using Single Parameter

Software University

```
class Collection<T> {
    public data: T[];
    constructor(...elements: T[]) { this.data = elements; }

    addElement(el: T) { this.data.push(el); }

    removElement(el: T) {
        let index = this.data.indexOf(el);
        if (index > -1) {
            this.data.splice(index, 1);
        }
    }

    reverseElements() { return this.data.reverse(); }

    showElements() { return this.data; }
}
```

Example: Generic Class Using Multiple Parameters


```
class UserInput<F, S> {
    public first: F;
    public second: S;
    constructor (f: F, s: S) {
        this.first = f;
        this.second = s;
    }

    showBoth() {
        return  `First: ${this.first}, second: ${this.second}`;
    }
}

let sample = new UserInput('Ten', 10);
let test = new UserInput(1, true);
console.log(sample.showBoth()); // First: Ten, second: 10
console.log(test.showBoth()); // First: 1, second: true
```

Example: Generic Class Implements Interface

Software University

```
interface ShowComponents<T, V> {
    print(key: T, value: V): string;
}

class Components<T, V> implements ShowComponents<T, V> {
    public key: T;
    public value: V;
    constructor(k: T, v: V) {
        this.key = k;
        this.value = v;
    }
    print(){
        return `Key: ${this.key} and value: ${this.value}`;
    }
}
let test: ShowComponents<string, string> = new Components('New', 'Test');
console.log(test.print('Test', 'Hello')); // Key: New and value: Test
```

Generic Type Constraints

Software University

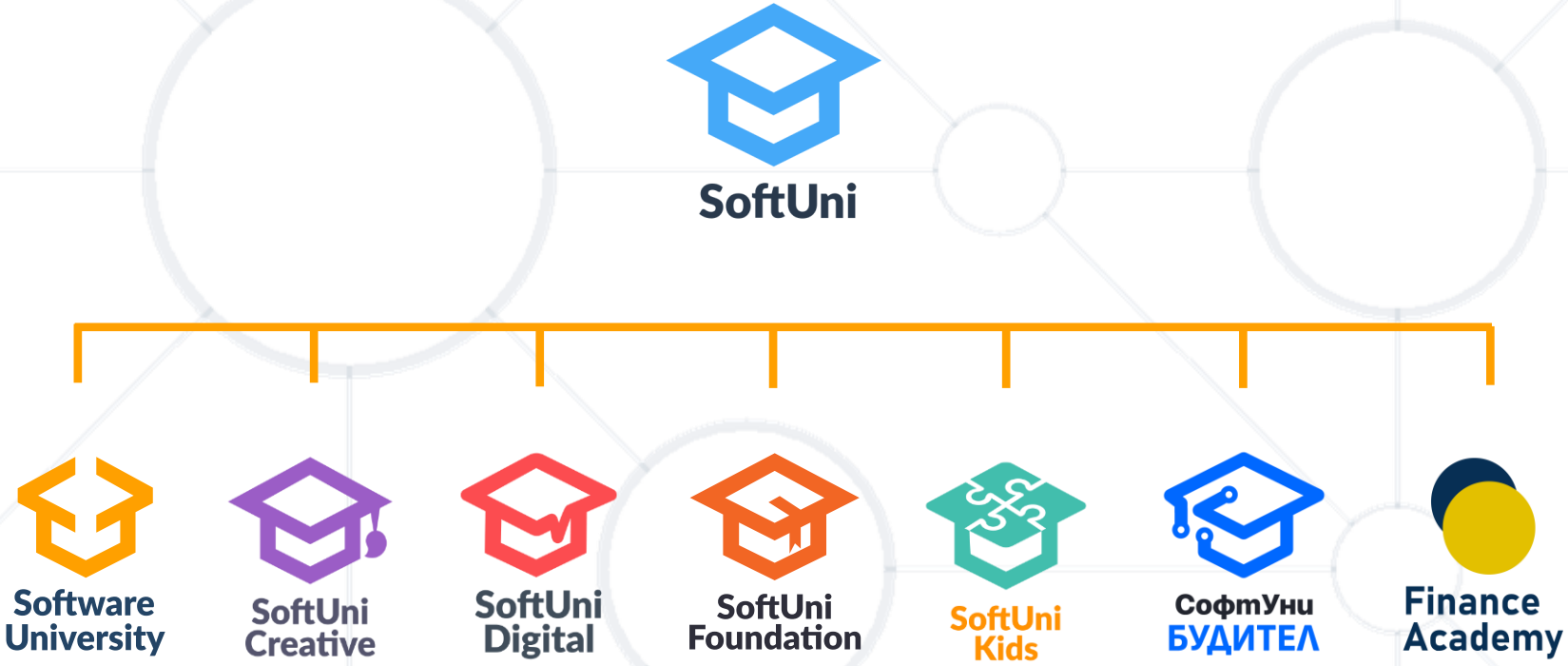
- In TypeScript we can make sure that sudden type variable **has** at least **some information** containing in it
- Constraints are enforced by **extends** keyword

```
function fullName<T extends { fName: string, lName: string }>(obj: T) {

      return `The full name is ${obj.fName} ${obj.lName}.`;
}

let output = fullName({fName: 'Svetoslv', lName: 'Dimitrov'});

console.log(output); // The full name is Svetoslav Dimitrov
```

Summary

- Generics are use to:
  - Abstract data types
  - Build reusable components
- We can use them in:
  - Functions
  - Classes - their properties and methods
  - Interfaces

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

Software University

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg