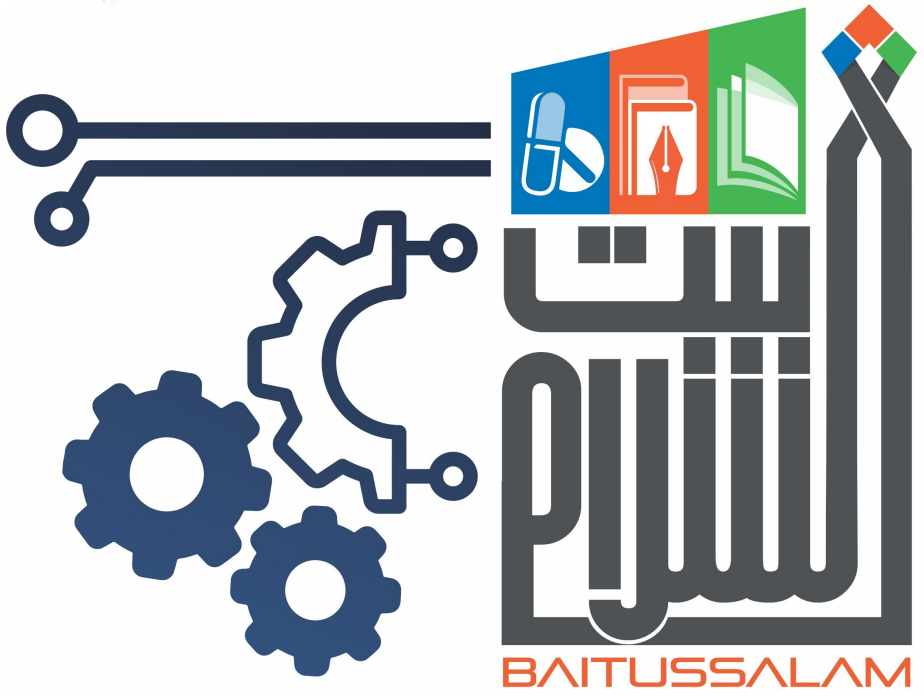

BAITUSSALAM
—TECH PARK—

Class Agenda

BAITUSSALAM
—TECH PARK—

Generic, Unit Testing, Javascript Frameworks

Generics

Code Reusability

Problem:

Without generics, you would need to write multiple versions of the same code for different types, leading to code duplication.

Solution:

Generics allow you to write a single function, class, or interface that works with any data type. This reduces redundancy and makes your code more maintainable.

Simple Generics Example

```
function identity<T>(arg: T): T {
  return arg
}

const str = identity('Hello')
console.log('str', str)

const num = identity(123)
console.log('num', num)
```

Generic Example #1

Define a function **firstElement** that takes an array of any type and returns the first element of that array.

By using generics, this function can handle arrays of any type while maintaining type safety.

Generic Example #2

Define a function **merge** that takes two objects as parameters and returns a new object combining both properties.

Generics will help ensure that the properties and types of the input objects are preserved and combined correctly.

```
const person = { name: 'John', age: 30 }
const address = { street: '123 Main St', city: 'New York' }

const result = merge(person, address)
```

Generic Constraints

- Constraints ensure that the generic type parameter satisfies certain conditions, such as having specific properties or methods.
- By constraining types, you can prevent errors that might occur if the generic type does not meet the expected criteria.

Constraints are defined using the extends keyword in the generic type parameter.

```
function example<T extends SomeType>(arg: T): void {
    // Function implementation
}
```

Constraints Example

```
interface Lengthwise {
  length: number
}

function logLength<T extends Lengthwise>(arg: T): void {
  console.log(arg.length)
}
```

- **T extends Lengthwise:** This constraint specifies that T must have a length property.
- The function **logLength** can now safely access the length property on arg.

Type Parameters in Generic Constraints

You can declare a type parameter that is constrained by another type parameter. For example, here we'd like to get a property from an object given its name. We'd like to ensure that we're not accidentally grabbing a property that does not exist on the obj, so we'll place a constraint between the two types:

```
function getProperty<Type, Key extends keyof Type>(obj: Type, key: Key) {
  return obj[key]
}

let x = { a: 1, b: 2, c: 3, d: 4 }

getProperty(x, 'a')
getProperty(x, 'm')
```

Class Types in Generics

Generic allows a class to operate with various types while providing type safety.

```
class GenericClass<T> {
  value: T

  constructor(value: T) {
    this.value = value
  }

  getValue(): T {
    return this.value
  }
}

const instance1 = new GenericClass('Smith')
const instance2 = new GenericClass(123456)
```

Class Types Generic Exercise

Read, Type, and Understand Generic in Classes example given in Typescript official Docs

Unit Testing

Unit testing is a software testing method where individual units or components of software are tested. The purpose is to validate that each software unit performs as expected.

Benefits

- Catches bugs early in the development cycle.
- It makes integration easier by ensuring each unit works correctly.

Types of testing

- **Unit Testing:** Testing individual units/components.
- **Integration Testing:** Testing combined parts of an application.
- **End to End to Testing:** E2E Testing, a software testing methodology that verifies the working order of a software product in a start-to-finish process

Test-Driven Development (TDD)

Test-Driven Development is a software development process where tests are written before writing the code that needs to be tested.

TDD Cycle:

Write a Test: Write a test for a new feature.

Run the Test: Run the test to see it fail (since the feature isn't implemented yet).

Write Code: Write the minimum code required to pass the test.

Run Tests: Run the tests again to ensure they pass.

Installing Dependencies

1. Create a JavaScript App with Vite
npm create vite@latest

2. Install Jest as dev dependency
npm install --save-dev jest

3. Add jest in script tag
"scripts": { "test": "jest" }

4. Run the test
npm test

Unit Testing Example

BAITUSSALAM
—TECH PARK—

Main Function file

utils > sum.js > ...

```
1  function sum(a, b) {
2    return a + b
3  }
4
5  module.exports = sum
6
```

Test file

filename.test.js

utils > sum.test.js > ...

```
1  const sum = require('./sum')
2
3  test('adds 1 + 2 to equal 3', () => {
4    expect(sum(1, 2)).toBe(3)
5  })
6
7  test('adds 2 + 5 to equal 7', () => {
8    expect(sum(2, 5)).toBe(7)
9  })
10
```

JavaScript Libraries and Framework

BAITUSSALAM
—TECH PARK—

Popular JS Library and Frameworks

1. jQuery
2. React
3. Vue
4. Angular
5. Meteor

Both frameworks and libraries are code written by someone else that is used to help solve common problems.

When you use a **library**, you are in charge of the flow of the application. You are choosing when and where to call the library.

When you use a **framework**, the framework is in charge of the flow. It provides some places for you to plug in your code, but it calls the code you plugged in as needed.

jQuery

Simplifies HTML DOM manipulation, event handling, and animation.

```
1  $(document).ready(function () {
2    $(".increment").on("click", function () {
3      console.log("increment clicked");
4    });
5
6    $(".decrement").on("click", function () {
7      console.log("decrement clicked");
8    });
9  });
10
```

React

- A library for building user interfaces.
- Component-based architecture.

React is categorized as a library because it provides a collection of tools, including components and functions, for use, without enforcing strict rules or structures for building the entire application.

Example of simple button component code in react

```
1  import React from 'react'
2
3  const Button = ({ text, onClick }) => {
4    return <button onClick={onClick}>{text}</button>
5  }
6
7  export default Button
8
```

Vue.js

A progressive framework for building user interfaces.

Hello World app in Vue.js

```vue
<template>
  <div id="app">
    <h1>Hello World</h1>
  </div>
</template>

<script>
export default {
  name: 'App',
};
</script>

<style>
/* Add any global styles here */
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

Pros and Cons of Using Libraries and Frameworks

Pros

Efficiency:
Saves time by providing pre-built functionalities.

Consistency:
Ensures consistent code patterns and practices.

Community Support:
Access to community-contributed plugins, extensions, and solutions.

Maintainability:
Well-documented and regularly updated.

Cons

Learning Curve:
Requires time to learn and understand.

Performance:
May add overhead and impact performance.

Dependency Management:
Reliance on third-party updates and support.

Overhead:
Can introduce unnecessary complexity for small projects.

The End

BAITUSSALAM
—TECH PARK—