# Basic Terminology

**React.Component**

class ParentClass extends React.Component {}

Using the extends keyword, you can allow the current component i.e Parent Class to access all the component's properties (React.Component properties), including the functions.

A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render() method, this method returns HTML.

**Example**

Create a Class component called Car

class Car extends React.Component {

render() {

return <h2>Hi, I am a Car!</h2>;

}

}

**Note :** **Render** means renew only an appropriate part of information on user’s screen when the element properties (props) are replaced by new ones or a component state (as set of props) changes in application. Thanks to the render method, we avoid reloading the whole web page, save time, and increase productivity.

**Rendering a Component**

Now your React application has a component called Car, which returns an <h2> element.

To use this component in your application, use similar syntax as normal HTML: <Car />

**Example**

Display the Car component in the "root" element:

ReactDOM.render(<Car />, document.getElementById('root'));

## Components in Files

React is all about re-using code, and it is recommended to split your components into separate files.

To do that, create a new file with a .js file extension and put the code inside it:

Note that the filename must start with an uppercase character.

**Example**

This is the new file, we named it "Car.js":

function Car() {

return <h2>Hi, I am a Car!</h2>;

}

export default Car;

To be able to use the Car component, you have to import the file in your application.

**Example**

Now we import the "Car.js" file in the application, and we can use the Car component as if it was created here.

import React from 'react';

import ReactDOM from 'react-dom';

import Car from './Car.js';

ReactDOM.render(<Car />, document.getElementById('root'));

Refer :

https://www.w3schools.com/react/react\_components.asp

Refer :

<https://softwareengineering.stackexchange.com/questions/178927/is-there-a-difference-between-a-component-and-a-module>

# Software Granularity Hierarchy

**Product** - application, library, service

**Module** - GUI, core logic, data, etc...

**Component** - purpose specific collection of objects

**Object** - collection of primitives

**Primitive** - numbers, functions, etc...

**Product**

Plain and simple, the Product is a working collection of connected functional modules.

**Module**

A module encapsulates a collection of components which all serve a common purpose as defined by the module requirements.  A module should be self-contained and complete, and while not really usable on its own.

**Component**

In terms of granularity the Component sits between the Module and the Object. The purpose of a component is to put together a collection of general purpose objects to form a purpose specific unit.

As the name implies, unlike the Module, the Component is not "self-contained", it is a part of a larger functional whole.

**Object**

Objects are the smaller building blocks of components. Objects are collections of primitives.

**Primitive**

Primitives are the smallest, simplest and lowest level of software development granularity. It's basically just integer and real numbers and functions/operators, although most languages have their own additional "first class citizens".

# ReactJS | Importing and Exporting

Refer : https://www.geeksforgeeks.org/reactjs-importing-exporting/

**Importing**

You may treat each React Component as a module itself.

Thus, it is possible to import/export React Components and is one of the basic operations to be performed. In React we use the keyword **import** and **from** to import a particular module or a named parameter. Let us now see the different ways we can use the import operation in React.

* **Importing default export:** Every module is said to have at most one default export. In order to import the default export from a file, we can use only the address and use the keyword import before it, or we can give a name to the import making the syntax as the following.

import GIVEN\_NAME from ADDRESS

* **Importing named values:** Every module can have several named parameters and in order to import one we should use the syntax as follows.

import { PARA\_NAME } from ADDRESS

* And similarly, for multiple such imports, we can use a comma to separate two-parameter names within the curly braces.
* **Importing a combination of Default Exports and Named Values:** In order to import a combination, we should use the following syntax.

import GIVEN\_NAME, { PARA\_NAME, ... } from ADDRESS

# Exporting

Now, importing is an operation that requires the permission of the module. Importing is possible only if the module or named property to be imported has been exported in its declaration. In React we use the keyword **export** to export a particular module or a named parameter or a combination. Let us now see the different ways we can use the export operation in React.

* **Exporting default export:** We have already learned that every module is said to have at most one default export. In order to export the default export from a file, we need to follow the syntax described below.

export default GIVEN\_NAME

* **Exporting named values:** Every module can have several named parameters and in order to export one we should use the syntax as follows.

export { PARA\_NAME }

* And similarly, for multiple such exports, we can use a comma to separate two-parameter names within the curly braces.

**What is the difference between export default and export?**

export default MyModule = () => console.log('foo')

import MyModule from './MyModule' //it works

import foobar from './MyModule' //it also works,

**export const exports with name :**

export const MyModule = () => console.log('foo')

import MyModule from './MyModule' //returns empty object since there is no default export

import { MyModule } from './MyModule' //here it works because by exporting without 'default' keyword we explicitly exported MyModule

* So, when you're exporting only one element from your module and you don't care of its name, use export default.
* If you want to export some specific element from your module and you do care of their names, use export const

**In Other Words:**

* Exports without a default tag are Named exports.
* Exports with the default tag are Default exports.

Named Exports

As the title suggests, named exports use the name of the function or class as their identifier.

export const MyComponent = () => {

...body

}

// or

export function MyComponent() {

...body

}

When you want to import a named component, you use the same name it was exported with. Names must be imported inside curly brackets.

import { MyComponent } from "./my-component"

Multiple Exports Per File

Named exports allow multiple exports in a single file.

export const MyFunction1 = () => {};

export const MyFunction2 = () => {};

export const MyFunction3 = () => {};

You can include whatever module you like in your import.

import { MyFunction1, MyFunction2, MyFunction3 } from "./my-components"

Default Exports

Default exports are created by including a default tag in the export. Usually, you see default exports happen at the bottom of a file, but it’s possible to define them when your component is declared.

const MyComponent = () => {}

export default MyComponent

// or

function MyComponent() {}

export default MyComponent

// or

export default function() {}

// or

export default () => {}

When importing a default export, you don’t use curly brackets.

import MyComponent from "./my-component"

Default Aliasing

When you import a default export, you can give it whatever name you want.

import WhateverNameIsBest from "./my-component"

Single Export

modules can only have one default export.



Combining

You can combine default and named exports in a single file.

export const NamedComponent1 = () => {};

export const NamedComponent2 = () => {};

const DefaultComponent = () => {};

export default DefaultComponent;

Importing is the same, named exports are in curly brackets, default is plaintext.

import DefaultComponent, { NamedComponent1, NamedComponent2 } from "./component"

React is a great example of a library that exports both default and named components.

import React, {useState, useEffect} from "react"

Conclusion

Choosing between named or default exports won’t make or break your application. However, it’s good to know the difference because it deepens your understanding and helps enable better overall architecture and code readability.

<https://stackoverflow.com/questions/36866718/when-to-use-export-default-and-export-constant-in-javascript>

<https://betterprogramming.pub/understanding-the-difference-between-named-and-default-exports-in-react-2d253ca9fc22>

Note :

But, React has introduced a new JSX transform with the release of React 17 which automatically transforms JSX without using React.createElement. This allows us to not import React, however, you'll need to import React to use Hooks and other exports that React provides. But if you have a simple component, you no longer need to import React.

## Default export (export default)

We use the `export default` syntax to do the default export in a file. It is possible to have one default export per file.

#### Example of export default

// We don't need to specify any name for default export export default function(x) {

return x \* x \* x;}

When we import from default export we can specify any name we like. Following is an example:

// We can specify any name we likeimport OrderComponent from "./DefaultExportOrder";

## Named export (export)

We use the `export <name>` syntax to do the named export in a file. It is possible to have multiple named exports per file.

export function cube(x) {

return x \* x \* x;}

When we import we need to specify the exact name surrounded in braces like below:

### Importing single export:

// ex. importing single export:import { cube } from "./DefaultExportOrder";

### Importing multiple export:

// ex. importing multiple exports:import { getPaymentOptions, getOrderForm } from "./Subscription";

### Giving a different name:

// ex. giving a named import a different name by using "as":import { getPaymentOptions as Payments } from "./Subscription";

#### Default export along with named imports:

# It is possible to use a default export along with named imports in the same statement:import React, { useEffect, useRef } from 'react';

Note :

https://medium.com/nerd-for-tech/how-imports-work-in-react-121e3741fb70

Note :

React and ReactDOM are default exports