Week 6

Context

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- In a typical React application, data is passed top-down (parent to child) via props.
- Context provides a way to share values between components without having to explicitly pass a prop through every level of the tree.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { createContext, useContext } from "react";

const Context = createContext("abc");

export function Main() {
  const value = "My Context Value";

  return (
    <Context.Provider value={value}>
      <MyComponent />
    </Context.Provider>
  );
}

function MyComponent() {
  const value = useContext(Context);

  return <span>{value}</span>;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyComponent/>);
```

By using Props

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Name(props) {
  return <h2>My name { props.name }!</h2>; }
```

```
function Display() {
  return (
    <>
      <Name name="Niranjan" />
    </>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Display />);
```

React Fragments

- Allow you to wrap or group multiple elements without adding an extra node to the DOM.
- This can be useful when rendering multiple child elements/components in a single parent component.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function Column() {
  return (
    <React.Fragment>
      <h1>JSS</h1>
      <h2>Polytechnic</h2>
    </React.Fragment>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Column/>);
```

React Higher-Order Component

- The React Higher-Order Components is an advanced technique that takes a component and returns a new component. It is used for reusing component logic.

```
function sub (x, y) {
  return x - y  }
function higherOrderComp(x, subReference) {
  return subReference(x, 10)  }
higherOrderComp(50, sub)
```

React Router

- React Router is **a standard library for routing in React**.
- It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL

- **Route params** are parameters whose values are set dynamically in a page's URL.
- This allows a route to render the same component while passing that component the dynamic portion of the URL, so that it can change its data based on the parameter.
- Examples include things like IDs for products, books, movies, users etc

```
<Route exact path="/movie/:id" component={MovieDetailsContainer} />
```

React router key components

- **routers, like <BrowserRouter> and <HashRouter>**

  <BrowserRouter>: It is used for handling the dynamic URL.

  <HashRouter>: It is used for handling the static request.

```
function App() {
  return <h1>Hello React Router</h1>;
}

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById("root")
);
```

- **route matchers, like <Route> and <Switch>**

```
<div>
      <Switch>
        { /* If the current URL is /contact, this route is rendered */
}
        <Route path="/contact">
          <AllContacts />
        </Route>
        { /*If none of the previous routes render anything, this route acts as a fallback */ }
```

```
    <Route path="/">
      <Home />
    </Route>
  </Switch>
</div>
```

- **navigation, like <Link>, <NavLink>, and <Redirect>**
  i. React Router provides a <Link> component to create links in your application. Wherever you render a <Link>, an anchor (<a>) will be rendered in your HTML document.

  Syntax: <Link to="/">Home</Link>
  ```
  // <a href="/">Home</a>
  ```

  ii. The <NavLink> is a special type of <Link> that can style itself as "active" when its to prop matches the current location.

  Syntax:
  ```
  <NavLink to="/react" activeClassName="hurray">
    React
  </NavLink>
  ```

  iii. Any time that you want to force navigation, you can render a <Redirect>. When a <Redirect> renders, it will navigate using its to prop.

  Syntax: <Redirect to="/login" />

Add React Router

- To add React Router in your application, run this in the terminal from the root directory of the application:

Command: `npm i -D react-router-dom`

Create a folder name called pages. Within a pages create following files.

Index.js

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
```

```
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Blogs.js

```
const Blogs = () => {
    return <h1>Blog Articles</h1>;
  };

  export default Blogs;
```

Contact.js

```
const Contact = () => {
    return <h1>Contact Me</h1>;
  };

  export default Contact;
```

Home.js

```
const Home = () => {
    return <h1>Home</h1>;
  };
   export default Home;
```

Layout.js

```
import { Outlet, Link } from "react-router-dom";

const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />
    </>
  )
};

export default Layout;
```

NoPage.js

```
const NoPage = () => {
    return <h1>404</h1>;
  };

  export default NoPage;
```

React Hooks – Introduction

- Hooks (Receive or Steal) are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class.

useState

- The React useState Hook allows us to track state in a function component.
- State generally refers to data or properties that need to be tracking in an application.
- To use the useState Hook, we first need to import it into our component.

Syntax: import { useState } from "react";

- We initialize our state by calling useState in our function component.
- useState accepts an initial state and returns two values:

i. The current state.

ii. A function that updates the state.

Initialization

```
import { useState } from "react";
function FavoriteColor() {
  const [color, setColor] = useState("");
}
```

Example for update and Read state

```
import { useState } from "react";
import ReactDOM from "react-dom/client";
function FavoriteColor() {
  const [color, setColor] = useState("red");
  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
    </>
  )}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FavoriteColor />);
```

useEffect:

- The useEffect Hook allows you to perform side effects in your components.
- Some examples of side effects are: fetching data, directly updating the DOM, and timers.
- useEffect accepts two arguments. The second argument is optional.

Syntax: useEffect(<function>, <dependency>)

Example to set timer:

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1>I have rendered {count} times!</h1>;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Timer />);
```

Example to clear timer:

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

function Timer() {
  const [count, setCount] = useState(0);
  useEffect(() => {
    let timer = setTimeout(() => {
    setCount((count) => count + 1);
  }, 1000);

  return () => clearTimeout(timer)
  }, []);
  return <h1>I've rendered {count} times!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Timer />);
```

useContext

- React Context is a way to manage state globally.
- It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

Create Context: You must Import createContext and initialize it:

```
import { useState, createContext } from "react";
import ReactDOM from "react-dom/client";
const UserContext = createContext()
```

Context Provider: Wrap child components in the Context Provider and supply the state value.

useContext: To use the Context in a child component, we need to access it using the useContext Hook.

```
import { useState, createContext, useContext } from "react";
import ReactDOM from "react-dom/client";
const UserContext = createContext();

function Component1() {
  const [user, setUser] = useState("Jesse Hall");
  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 />
    </UserContext.Provider>
  );
}
function Component2() {
  return (
    <>
```

```
      <h1>Component 2</h1>
      <Component3 />
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Component1 />);
```

useReducer

- The useReducer Hook is similar to the useState Hook.
- It allows for custom state logic.
- If you find yourself keeping track of multiple pieces of state.

Syntax: useReducer(<reducer>, <initialState>)

- The reducer function contains your custom state logic and the initialStatecan be a simple value but generally will contain an object.

Example:

```
import React, { useReducer } from "react";

// Defining the initial state and the reducer
const initialState = 0;
const reducer = (state, action) => {
  switch (action) {
    case "add":
      return state + 1;
    case "subtract":
      return state - 1;
    case "reset":
      return 0;
    default:
      throw new Error("Unexpected action");
  }
};

const App = () => {
    // Initialising useReducer hook
  const [count, dispatch] = useReducer(reducer, initialState);
```

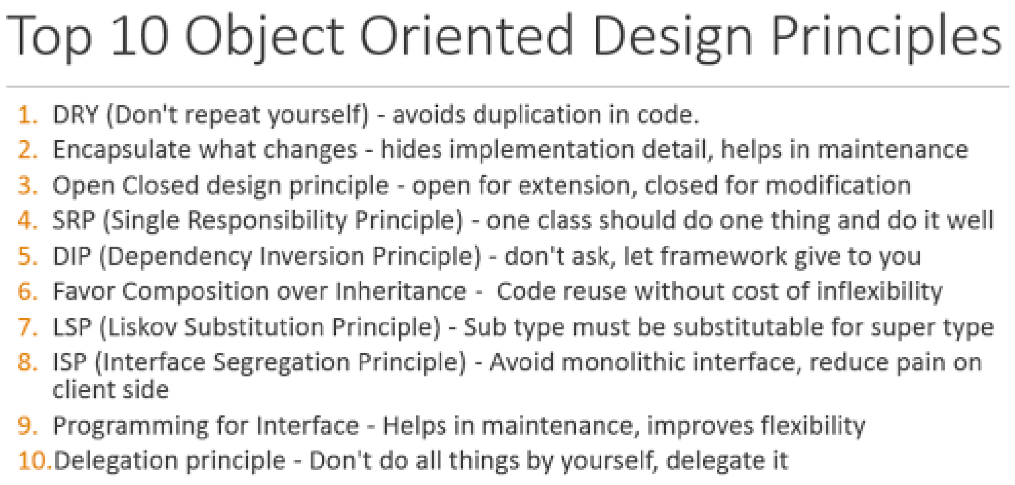
```
 return (
   <div>
     <h2>{count}</h2>
     <button onClick={() => dispatch("add")}>
       add
     </button>
     <button onClick={() => dispatch("subtract")}>
       subtract
     </button>
     <button onClick={() => dispatch("reset")}>
       reset
     </button>
   </div>
 );
};

export default App;
```

Custom Hooks

- Hooks are reusable functions.
- When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.
- Custom Hooks start with "use". Example: useFetch.

Object oriented concepts and designprinciples

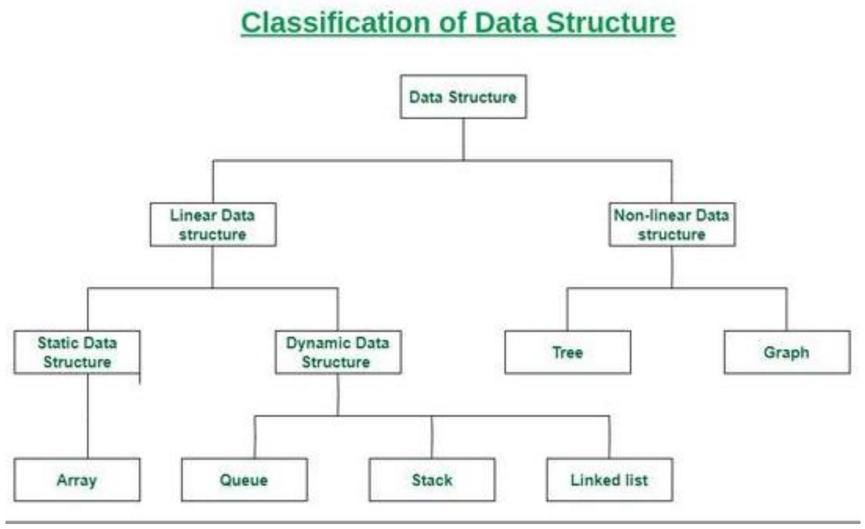


Top 10 Object Oriented Design Principles

1. DRY (Don't repeat yourself) - avoids duplication in code.
2. Encapsulate what changes - hides implementation detail, helps in maintenance
3. Open Closed design principle - open for extension, closed for modification
4. SRP (Single Responsibility Principle) - one class should do one thing and do it well
5. DIP (Dependency Inversion Principle) - don't ask, let framework give to you
6. Favor Composition over Inheritance - Code reuse without cost of inflexibility
7. LSP (Liskov Substitution Principle) - Sub type must be substitutable for super type
8. ISP (Interface Segregation Principle) - Avoid monolithic interface, reduce pain on client side
9. Programming for Interface - Helps in maintenance, improves flexibility
10. Delegation principle - Don't do all things by yourself, delegate it

Data Structure:

A data structure is **a specialized format for organizing, processing, retrieving and storing data**.



Database Concepts:

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).

- **Database Schema:** It is a design of the database.
- **Data Constraints:** Constraints are used to put some restriction on the table.
- **Data dictionary or Metadata:** Metadata is known as the data about the data
- **Database instance:** used to define the complete database environment and its components.
- **Query:** query is used to access data from the database.
- **Data manipulation:** manipulation can be done using three main operations that is Insertion, Deletion, and updation.
- **Data Engine:** used to create and manage various database queries.

Java and servlet basics

- Java is a popular programming language.
- Java is used to develop mobile apps, web apps, desktop apps, games and much more.

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Java Collections (List, Set, Map)

A **List** in java extends the collection interface and represent an sequenced or ordered group of elements. It can contain duplicate elements.

It also defines some additional methods which it inherits from Collection interface.

```
import java.util.*;
public class Main {
    public static void main(String args[]){
        List<String> mySubjects = new ArrayList<>();
        mySubjects.add("Java");
        mySubjects.add("Spring");
         System.out.println("My Subjects:");
        for(String subject : mySubjects){
         System.out.println(subject);
        }
      }
}
```

Set

- A set represents a group or collection of items. Set has a special property that is unique items, it can not contain a duplicate item or element. It extends the collection interface.

```
import java.util.*;
public class Main {
     public static void main(String args[]){
        Set<String> mySubjects = new HashSet<>();
         mySubjects.add("Java");
        mySubjects.add("Spring");
        System.out.println("My Subjects:");
        for(String subject : mySubjects){
         System.out.println(subject);
        }
      }
 }
```

Map

- It represents a group of special elements or objects. Every map element or object contains key and value pair. A map can't contain duplicate keys and one key can refer to at most one value.

```
import java.util.*;
public class Main {

   public static void main(String args[]){
      Map<Integer,String> mysubjects = new HashMap<Integer,String>();

      mysubjects.put(1,"Java");
      mysubjects.put(2,"Spring");
      for(Map.Entry subject : mysubjects.entrySet())
        System.out.println(subject.getKey()+" - "+subject.getValue());

   }
}
```

Thread:

- Threads allows a program to operate more efficiently by doing multiple things at the same time.
- Threads can be used to perform complicated tasks in the background without interrupting the main program.

Install java (latest stable version) and addenvironment variable

1. **Install and Setup java environment**

   Step 1: Download JDK

   @ http://www.oracle.com/technetwork/java/javase/downloads/index.html.

   Step 2: Install JDK

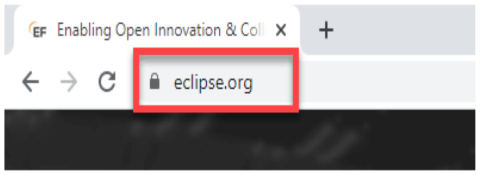
   Step 3: Include JDK's "bin" Directory in the PATH

   Step 4: Verify the JDK Installation

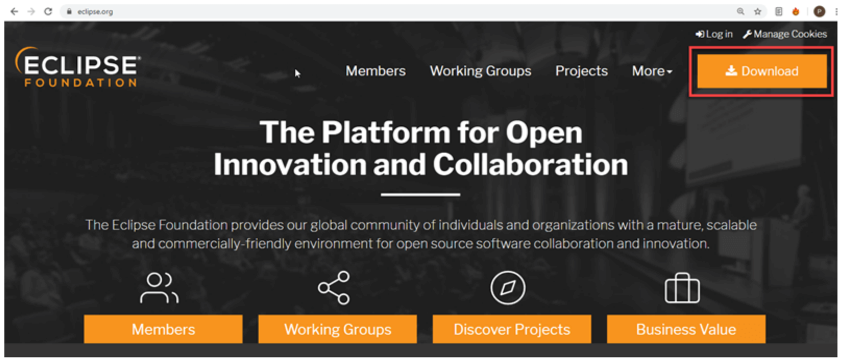
Install java editor (Eclipse for Enterprise Java) and configure workspace

Step 1) Installing Eclipse

Open your browser and type https://www.eclipse.org/



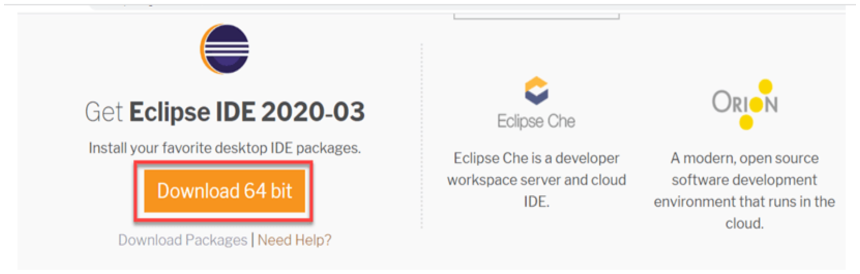
Step 2) Click on "Download" button.



The Platform for Open Innovation and Collaboration

The Eclipse Foundation provides our global community of individuals and organizations with a mature, scalable and commercially-friendly environment for open source software collaboration and innovation.

Step 3) Click on "Download 64 bit" button



Get Eclipse IDE 2020-03

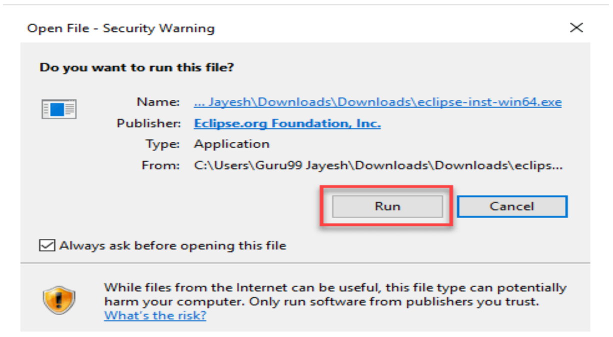
Install your favorite desktop IDE packages.

Eclipse Che is a developer workspace server and cloud IDE.

A modern, open source software development environment that runs in the cloud.

Step 4) Install Eclipse.

Click on Run button



Open File - Security Warning

Do you want to run this file?

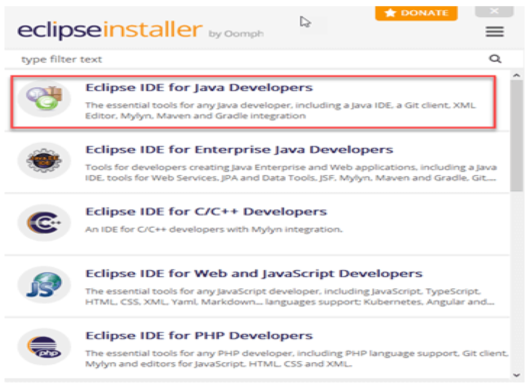
Name: ... Jayesh\Downloads\Downloads\eclipse-inst-win64.exe
Publisher: Eclipse.org Foundation, Inc.
Type: Application
From: C:\Users\Guru99 Jayesh\Downloads\Downloads\eclips...

Run Cancel

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Step 5) Click on “Eclipse IDE for Java Developers”



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

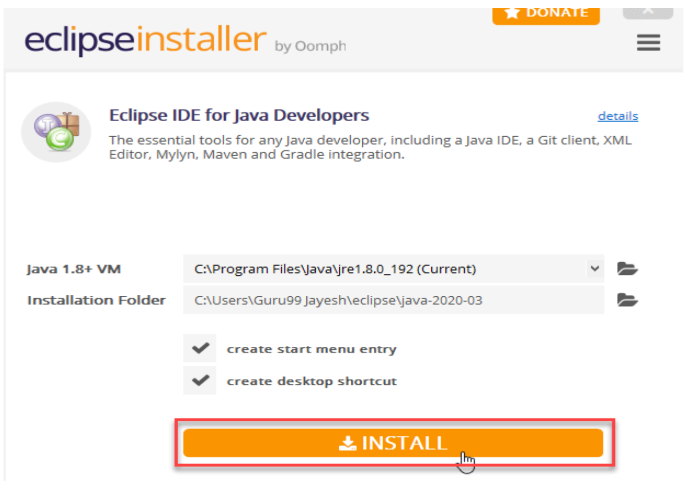
Eclipse IDE for Enterprise Java Developers
Tools for developers creating Java Enterprise and Web applications, including a Java IDE, tools for Web Services, JPA and Data Tools, JSF, Mylyn, Maven and Gradle, Git,...

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.

Eclipse IDE for Web and JavaScript Developers
The essential tools for any JavaScript developer, including JavaScript, TypeScript, HTML, CSS, XML, Yaml, Markdown... languages support; Kubernetes, Angular and...

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.

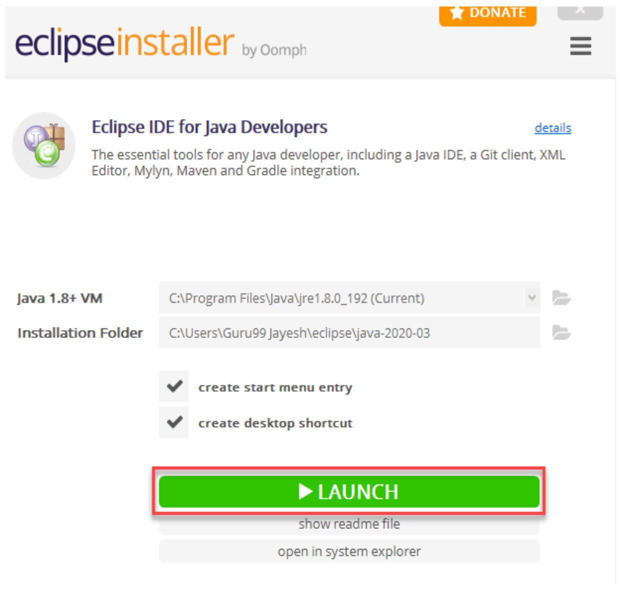
Step 6) Click on “INSTALL” button



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_192 (Current)
Installation Folder: C:\Users\Guru99 Jayesh\eclipse\java-2020-03
✓ create start menu entry
✓ create desktop shortcut

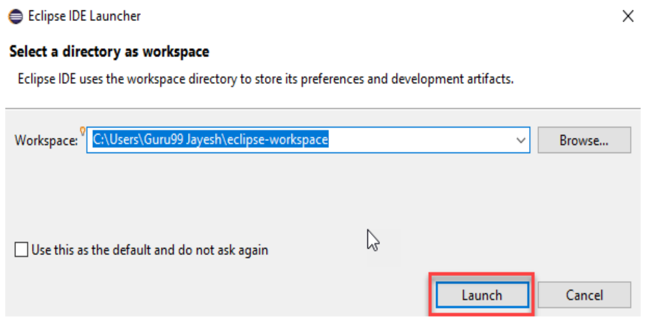
Step 7) Click on “LAUNCH” button.



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_192 (Current)
Installation Folder: C:\Users\Guru99 Jayesh\eclipse\java-2020-03
✓ create start menu entry
✓ create desktop shortcut

Step 8) Click on “Launch” button.



Eclipse IDE Launcher

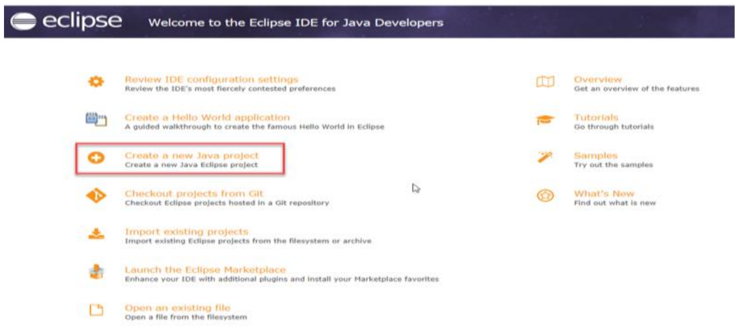
Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\Guru99 Jayesh\eclipse-workspace

☐ Use this as the default and do not ask again

Step 9) Click on “Create a new Java project” link.



Welcome to the Eclipse IDE for Java Developers

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse

Create a new Java project
Create a new Java Eclipse project

Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or archive

Launch the Eclipse Marketplace
Enhance your IDE with additional plugins and install your Marketplace favorites

Open an existing file
Open a file from the filesystem

Overview
Get an overview of the features

Tutorials
Go through tutorials

Samples
Try out the samples

What's New
Find out what is new

Step 10) Create a new Java Project

1. Write project name.
2. Click on “Finish button”.

Step 11) Create Java Package.

1. Goto “src”.
2. Click on “New”.
3. Click on “Package”.

Step 12) Writing package name.

1. Write name of the package
2. Click on Finish button.

Step 13) Creating Java Class

1. Click on package you have created.
2. Click on “New”.
3. Click on “Class”.

Step 14) Defining Java Class.

1. Write class name
2. Click on “public static void main (String[] args)” checkbox.
3. Click on “Finish” button.

Step 15) Click on “Run” button.

Redux

Redux is used by ReactJS for building the user interface and to manage the application state. The official React binding for Redux is React Redux which is used to read data from a Redux Store, and dispatch Actions to the Store to update data. It also helps the apps to scale.

State Management with Redux

State management is essentially a way to facilitate communication and sharing of data across components. It creates a tangible data structure to represent the state of your app that you can read from and write to.

Build single page application – like shopping Cart