National Computer Institute
7236996305,9453055513

Java Programming Notes

1. Introduction to Java

- Java is a **high-level, object-oriented programming language**.
- Developed by **James Gosling** at **Sun Microsystems** (now owned by **Oracle Corporation**).
- Follows **WORA (Write Once, Run anywhere)** using the **Java Virtual Machine (JVM)**.

2. Features of Java

- Platform Independent
- Object-Oriented
- Secure
- Robust (strong memory management)
- Multithreaded
- Distributed
- High Performance (with JIT compiler)

3. Basic Structure of a Java Program

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Key Points:

- `class` → Defines a class
- `main()` → Entry point of program
- `System.out.println()` → Prints output

4. Data Types

Primitive Data Types:

- `int` (Integer)
- `float`

- `double`
- `char`
- `boolean`
- `byte`, `short`, `long`

Non-Primitive:

- Arrays
- Strings
- Classes
- Interfaces

5. Variables

```
int age = 20;
float price = 99.99f;
char grade = 'A';
```

6. Operators

- Arithmetic: `+ - * / %`
- Relational: `== != > <`
- Logical: `&& || !`
- Assignment: `= += -=`

7. Control Statements

If-Else:

```
if (age > 18) {
    System.out.println("Adult");
} else {
    System.out.println("Minor");
}
```

Loops:

- `for`
- `while`

- `do-while`

8. Object-Oriented Concepts

1. Class & Object

```
class Car {
    String color;
}

Car c1 = new Car();
```

2. Encapsulation

- Wrapping data + methods together

3. Inheritance

```
class Animal {
    void eat() {}
}
class Dog extends Animal {}
```

4. Polymorphism

- Method Overloading
- Method Overriding

5. Abstraction

- Using `abstract` classes and interfaces

---

9. Exception Handling

```
try {
    int x = 10 / 0;
} catch (Exception e) {
    System.out.println("Error");
}
```

---

10. Arrays

```
int arr[] = {1, 2, 3, 4};
System.out.println(arr[0]);
```

---

11. Strings

```
String name = "Java";
System.out.println(name.length());
```

---

12. Multithreading

- Java supports multiple threads

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running");
    }
}
```

---

13. Java Collections Framework

- `ArrayList`
- `HashMap`
- `HashSet`
- `LinkedList`

---

14. JVM, JRE, JDK

- **JVM**: Runs Java bytecode
- **JRE**: JVM + libraries
- **JDK**: Development tools + JRE

---

15. Advantages of Java

- Easy to learn
- Secure
- Portable
- Rich API

---

16. Applications of Java

- Web Applications
- Mobile Apps (Android)
- Desktop Applications
- Enterprise Systems

National Computer Institute
7236996305,9453055513

Python Programming Notes

1. Introduction to Python

- Python is a **high-level, interpreted programming language**.
- Created by **Guido van Rossum** in 1991.
- Known for **simple syntax and readability**.

---

2. Features of Python

- Easy to learn and use
- Interpreted language
- Dynamically typed
- Object-oriented
- Large standard library
- Cross-platform

---

3. Basic Python Program

```
print("Hello, World!")
```

- No need for `main()` function
- No semicolons required

---

4. Variables and Data Types

Variables:

```
x = 10
name = "John"
price = 99.5
```

Data Types:

- `int`
- `float`

- `str`
- `bool`
- `list`
- `tuple`
- `set`
- `dict`

---

5. Operators

- Arithmetic: `+ - * / % // **`
- Comparison: `== != > <`
- Logical: `and or not`
- Assignment: `= += -=`

---

6. Control Statements

If-Else:

```
age = 18
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

Loops:

```
# for loop
for i in range(5):
    print(i)

# while loop
i = 0
while i < 5:
    print(i)
    i += 1
```

---

7. Functions

```
def greet(name):
    return "Hello " + name

print(greet("Aman"))
```

---

8. Lists

```
numbers = [1, 2, 3, 4]
numbers.append(5)
```

---

9. Tuples

```
t = (1, 2, 3)
```

---

10. Dictionaries

```
student = {"name": "Aman", "age": 20}
print(student["name"])
```

---

11. Strings

```
text = "Python"
print(len(text))
print(text.lower())
```

---

12. Exception Handling

```
try:
    x = 10 / 0
except:
    print("Error occurred")
```

---

13. Object-Oriented Programming

```
class Car:
    def __init__(self, color):
        self.color = color

c1 = Car("Red")
```

Concepts:

- Class & Object
- Inheritance
- Polymorphism

- Encapsulation

---

14. Modules and Packages

```
import math
print(math.sqrt(16))
```

---

15. File Handling

```
file = open("test.txt", "r")
print(file.read())
file.close()
```

---

16. Advantages of Python

- Simple syntax
- Large community support
- Used in AI, Data Science, Web Development

---

17. Applications of Python

- Web Development
- Data Science & Machine Learning
- Automation/Scripting
- Game Development

CSS (Cascading Style Sheets) – Quick Notes

📌 What is CSS?

CSS is used to style and layout web pages—for example, to change colors, fonts, spacing, and positioning.

□ Basic Syntax

```
selector {
  property: value;
}
```

Example:

```
p {
  color: blue;
  font-size: 16px;
}
```

🎯 Types of CSS

1. **Inline CSS**

```
<p style="color:red;">Hello</p>
```

2. **Internal CSS**

```
<style>
  p { color: red; }
</style>
```

3. **External CSS** (Best practice)

```
<link rel="stylesheet" href="styles.css">
```

🔍 Selectors

- **Element selector**

```
h1 { color: green; }
```

- **Class selector**

```
.box { border: 1px solid black; }
```

- **ID selector**

```
#header { background: gray; }
```

- **Group selector**

```
h1, p { color: black; }
```

🎨 Colors & Backgrounds

```
body {
  background-color: #f4f4f4;
  color: #333;
}
```

🎁 Box Model

Every element consists of:

- Content
- Padding
- Border
- Margin

```
div {
  margin: 10px;
  padding: 15px;
  border: 1px solid black;
}
```

📐 Layout (Important)

Flexbox

```
.container {
  display: flex;
```

```
  justify-content: center;
  align-items: center;
}
```

Grid

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

🔤 Fonts & Text

```
p {
  font-family: Arial, sans-serif;
  font-size: 18px;
  text-align: center;
}
```

🖱️ Hover Effects

```
button:hover {
  background-color: blue;
  color: white;
}
```

📱 Responsive Design

```
@media (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

✅ Best Practices

- Use external CSS files
- Keep code clean and reusable
- Use meaningful class names
- Avoid too many IDs
- Follow mobile-first design

CODING

1. Master Layout Systems

Flexbox (1D layout)

Used for aligning items in a row or column.

```
.container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

Grid (2D layout)

Powerful for full page layouts.

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
}
```

☞ Advanced tip: Combine **Grid + Flexbox** for real-world layouts.

◆ 2. Responsive Design (Beyond Basics)

Use **media queries + fluid units**:

```
body {
  font-size: clamp(16px, 2vw, 24px);
}

@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
}
```

☞ Learn:

- `clamp(),min(),max()`

- Mobile-first design

◆ 3. CSS Variables (Custom Properties)

```
:root {
  --main-color: #3498db;
}

button {
  background: var(--main-color);
}
```

☞ Advanced use:

- Theme switching (dark/light mode)
- Dynamic styling with JS

◆ 4. Advanced Selectors

```
/* Select every 2nd item */
li:nth-child(2n) {
  color: red;
}

/* Select parent if it has child */
div:has(img) {
  border: 2px solid blue;
}
```

◆ 5. Animations & Transitions

```
.box {
  transition: transform 0.3s ease;
}

.box:hover {
  transform: scale(1.2);
}
```

Keyframes:

```
@keyframes slide {
  from { transform: translateX(0); }
```

```
  to { transform: translateX(100px); }
}
```

◆ 6. Pseudo-elements & Effects

```
button::before {
  content: "🔥";
}
```

Create complex UI without extra HTML.

◆ 7. Modern CSS Features (Important 🚀)

Glassmorphism

```
.glass {
  backdrop-filter: blur(10px);
  background: rgba(255,255,255,0.2);
}
```

Container Queries (next-level responsive)

```
@container (min-width: 400px) {
  .card {
    flex-direction: row;
  }
}
```

◆ 8. Performance & Clean Code

- Avoid deep nesting
- Use reusable classes (BEM naming)
- Minimize reflows

◆ 9. Real Project Practice

Build:

- Dashboard UI
- Responsive portfolio
- E-commerce layout

◆ 10. Tools Professionals Use

- Sass (variables, nesting)
- Tailwind CSS (fast UI building)
- Bootstrap

1. Core Concept of 3D in CSS

To create 3D, you mainly use:

- `transform`
- `perspective`
- `transform-style`
- `backface-visibility`

◆ 2. Basic 3D Transform

```
.box {
  transform: rotateX(45deg) rotateY(45deg);
}
```

☞ This rotates the element in 3D space.

◆ 3. Add Perspective (VERY IMPORTANT)

Without perspective, 3D looks flat.

```
.container {
  perspective: 1000px;
}
```

☞ Lower value = more dramatic 3D
☞ Higher value = subtle effect

◆ 4. Create a 3D Cube 🔥

HTML

```
<div class="container">
  <div class="cube">
    <div class="face front"></div>
    <div class="face back"></div>
    <div class="face left"></div>
    <div class="face right"></div>
    <div class="face top"></div>
    <div class="face bottom"></div>
  </div>
</div>
```

CSS

```
.container {
  perspective: 800px;
}

.cube {
  width: 200px;
  height: 200px;
  position: relative;
  transform-style: preserve-3d;
  transform: rotateX(30deg) rotateY(30deg);
}

.face {
  position: absolute;
  width: 200px;
  height: 200px;
  background: rgba(0, 150, 255, 0.7);
  border: 2px solid #fff;
}

/* Position each face */
.front  { transform: translateZ(100px); }
.back   { transform: rotateY(180deg) translateZ(100px); }
.left   { transform: rotateY(-90deg) translateZ(100px); }
.right  { transform: rotateY(90deg) translateZ(100px); }
.top    { transform: rotateX(90deg) translateZ(100px); }
.bottom { transform: rotateX(-90deg) translateZ(100px); }
```

◆ 5. Add Animation (Spinning Cube)

```
@keyframes spin {
  from { transform: rotateX(0) rotateY(0); }
```

```
  to   { transform: rotateX(360deg) rotateY(360deg); }
}

.cube {
  animation: spin 5s infinite linear;
}
```

◆ 6. Flip Card Effect (Popular UI)

```
.card {
  perspective: 1000px;
}

.inner {
  transform-style: preserve-3d;
  transition: transform 0.6s;
}

.card:hover .inner {
  transform: rotateY(180deg);
}

.front, .back {
  backface-visibility: hidden;
  position: absolute;
}

.back {
  transform: rotateY(180deg);
}
```

☞ Used in:

- Product cards
- Login/signup panels
- Interactive UI

◆ 7. Advanced Tricks 🚀

Parallax 3D Effect

```
.layer {
  transform: translateZ(-200px) scale(2);
}
```

Mouse-based 3D Tilt (JS + CSS)

Combine with JavaScript to rotate based on cursor.

◆ 8. Common Mistakes

- Forgetting `perspective`
- Not using `transform-style: preserve-3d`
- Ignoring `backface-visibility`
- Overusing effects → hurts performance

◆ 9. Real Use Cases

- Landing pages (modern UI)
- Game-like interfaces
- Interactive portfolios
- Product showcases

□ Pro Tip

CSS 3D is powerful, but for **heavy 3D graphics**, developers switch to:

- Three.js
- WebGL

1. Basic 2D Transformations in CSS

CSS provides a `transform` property for 2D effects.

Translate (Move an element)

```
.box {
  transform: translate(50px, 100px);
}
```

Moves the element **right 50px** and **down 100px**.

Rotate

```
.box {
  transform: rotate(45deg);
}
```

Rotates the element **45 degrees clockwise**.

Scale (Resize)

```
.box {
  transform: scale(1.5, 1.5);
}
```

Makes the element **1.5× bigger**.

Skew (Tilt)

```
.box {
  transform: skew(20deg, 10deg);
}
```

Tilts the element along X and Y axes.

Combine Multiple Transforms

```
.box {
  transform: translate(50px, 50px) rotate(30deg) scale(1.2);
}
```

◆ 2. Example (Simple Box Animation)

```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
  width: 100px;
  height: 100px;
  background: blue;
  transition: 0.5s;
}

.box:hover {
  transform: rotate(45deg) scale(1.2);
}
</style>
</head>
<body>

<div class="box"></div>

</body>
</html>
```

☞ When you hover, the box rotates and grows.

◆ 3. 2D Layout Techniques (Very Important)

2D CSS isn't just transforms—it also includes layout systems:

Flexbox (1D but commonly used)

```
.container {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

CSS Grid (True 2D Layout)

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 10px;
}
```

◆ 4. When to Use 2D CSS

Use it for:

- Animations (buttons, cards, hover effects)
- UI design (menus, modals)
- Layouts (grid systems)
- Interactive elements

◆ 5. Pro Tip

Add smooth animations with `transition`:

```
.box {
  transition: transform 0.3s ease;
}
```

What is AngularJS?

AngularJS is a front-end JavaScript framework developed by Google to build dynamic single-page web applications (SPAs). It extends HTML with additional attributes and makes it more expressive.

□ Key Features

- **Two-way Data Binding**
  Automatically syncs data between model and view.
- **MVC Architecture**
  Separates application into:
  - Model (data)
  - View (UI)
  - Controller (logic)
- **Directives**
  Custom HTML attributes (e.g., `ng-model`, `ng-bind`, `ng-repeat`).
- **Dependency Injection (DI)**
  Automatically injects required services into components.
- **Templates**
  HTML enhanced with AngularJS expressions (`{{ }}`).

□ Architecture Overview

1. Module

The starting point of an AngularJS app:

```
var app = angular.module('myApp', []);
```

2. Controller

Controls data and logic:

```
app.controller('myCtrl', function($scope) {
  $scope.name = "John";
});
```

3. Scope

- Acts as a bridge between controller and view.
- Holds application data.

🔧 Important Directives

Directive	Description
`ng-app`	Initializes AngularJS app
`ng-model`	Binds input fields to data
`ng-bind`	Binds data to HTML
`ng-repeat`	Repeats elements
`ng-if`	Conditional rendering
`ng-click`	Handles click events

🔄 Data Binding Example

```
<div ng-app="">
  <input type="text" ng-model="name">
  <p>Hello {{name}}</p>
</div>
```

☞ Updates automatically when input changes.

□ Filters

Used to format data:

```
{{ price | currency }}
{{ name | uppercase }}
```

Services

Reusable components for common tasks:

- `$http` → API calls
- `$timeout` → Delay execution
- `$filter` → Data formatting

Example:

```
app.controller('myCtrl', function($scope, $http) {
  $http.get("data.json").then(function(response) {
    $scope.data = response.data;
  });
});
```

Routing (ngRoute)

Used for SPA navigation:

```
app.config(function($routeProvider) {
  $routeProvider
    .when('/home', {
      templateUrl: 'home.html'
    })
    .otherwise({
      redirectTo: '/'
    });
});
```

Limitations

- Slower performance for large apps
- Complex for large-scale architecture
- No longer actively developed (superseded by modern Angular)

AngularJS vs Modern Angular

Feature	AngularJS	Modern Angular

Feature	AngularJS	Modern Angular
Language	JavaScript	TypeScript
Architecture	MVC	Component-based
Performance	Slower	Faster
Mobile support	Limited	Strong

Coding

1. Basic AngularJS App

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="myApp">

<div ng-controller="myCtrl">
  <h2>Hello {{name}}</h2>
</div>

<script>
var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {
  $scope.name = "AngularJS";
});
</script>

</body>
</html>
```

2. Two-Way Data Binding

```
<div ng-app="">
  <input type="text" ng-model="username">
  <h3>Welcome {{username}}</h3>
</div>
```

3. Using `ng-repeat` (Loop)

```
<div ng-app="myApp" ng-controller="listCtrl">
 <ul>
  <li ng-repeat="item in items">{{item}}</li>
 </ul>
</div>

<script>
angular.module('myApp', [])
.controller('listCtrl', function($scope) {
 $scope.items = ["Apple", "Banana", "Mango"];
});
</script>
```

4. Event Handling (`ng-click`)

```
<div ng-app="myApp" ng-controller="clickCtrl">
 <button ng-click="count = count + 1">Click Me</button>
 <p>Count: {{count}}</p>
</div>

<script>
angular.module('myApp', [])
.controller('clickCtrl', function($scope) {
 $scope.count = 0;
});
</script>
```

5. API Call using `$http`

```
<div ng-app="myApp" ng-controller="apiCtrl">
  <ul>
    <li ng-repeat="user in users">{{user.name}}</li>
  </ul>
</div>

<script>
angular.module('myApp', [])
.controller('apiCtrl', function($scope, $http) {
  $http.get("https://jsonplaceholder.typicode.com/users")
  .then(function(response) {
    $scope.users = response.data;
  });
});
</script>
```

6 Routing Example

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <script src="angular.min.js"></script>
  <script src="angular-route.min.js"></script>
</head>

<body>
  <a href="#!home">Home</a>
  <a href="#!about">About</a>

  <div ng-view></div>

<script>
var app = angular.module('myApp', ['ngRoute']);

app.config(function($routeProvider) {
  $routeProvider
  .when("/home", {
    template: "<h1>Home Page</h1>"
  })
  .when("/about", {
    template: "<h1>About Page</h1>"
  });
});
```

```
</script>

</body>
</html>
```

7. Custom Directive

```
angular.module('myApp', [])
.directive('myMessage', function() {
 return {
  template: "<h2>Hello from Directive!</h2>"
 };
});
```

Usage:

<my-message></my-message>

8. Filter Example

```
<div ng-app="" ng-init="price=100">
 <p>{{price | currency}}</p>
 <p>{{'angular' | uppercase}}</p>
</div>
```

End

Understanding in Class

(only notes)

National Computer Institute

7236996305,9453055513

Mini Project:----------

```
<!DOCTYPE html>
<html>
<head>
 <title>AngularJS Todo App</title>
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

 <style>
  body {
   font-family: Arial;
   margin: 30px;
  }

  .done {
   text-decoration: line-through;
   color: gray;
  }

  button {
   margin-left: 10px;
  }
 </style>
</head>

<body ng-app="todoApp" ng-controller="todoCtrl">

 <h2>📝 My To-Do List</h2>

 <!-- Input -->
 <input type="text" ng-model="newTask" placeholder="Enter task">
 <button ng-click="addTask()">Add</button>

  <ul>
   <li ng-repeat="task in tasks">

    <!-- Mark done -->
    <span ng-class="{'done': task.done}" ng-click="toggleTask(task)">
     {{task.name}}
    </span>
```

```
    <!-- Delete task -->
    <button ng-click="deleteTask($index)">□</button>
  </li>
 </ul>

<script>
var app = angular.module("todoApp", []);

app.controller("todoCtrl", function($scope) {

 // Task list
 $scope.tasks = [
  {name: "Learn AngularJS", done: false},
  {name: "Build project", done: false}
 ];

 // Add task
 $scope.addTask = function() {
  if ($scope.newTask) {
   $scope.tasks.push({name: $scope.newTask, done: false});
   $scope.newTask = "";
  }
 };

 // Toggle done/undone
 $scope.toggleTask = function(task) {
  task.done = !task.done;
 };

 // Delete task
 $scope.deleteTask = function(index) {
  $scope.tasks.splice(index, 1);
 };

});
</script>

</body>
</html>
```