Software Development Concepts

Fundamental Concepts and Paradigms in the Software Engineering Profession

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

1. The 4 Skills of the Software Engineers
2. Fundamental Software Engineering Concepts
   - Math Concepts in Software Development
   - Object-Oriented Programming (OOP)
   - Functional Programming (FP)
   - Data Structures and Algorithms
   - Component-Based Development and Event-Driven Programming
3. Software Architectures, Front-End and Back-End

The 4 Skills of Software Engineers

Skills of the Software Engineers

- 4 main **groups of technical skills**
  - **Coding** skills – 20%
  - **Algorithmic** thinking – 30%
  - Fundamental software development **concepts** – 25%
  - Programming languages and software **technologies** – 25%

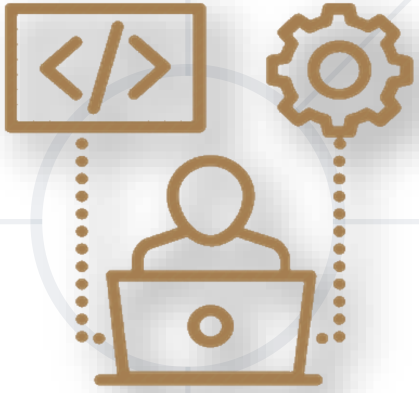
Skill #1: Coding (20%)

- The skill to **write code**
  - Working with **commands**, IDE, variables, data and **calculations**, **conditional** statements, **loops**
  - Using **functions** (or methods) and **objects**
  - Working with **data structures** (arrays, lists, maps and others), libraries and APIs
- **Courses** at SoftUni: softuni.bg/curriculum
  - Programming Basics, Programming Fundamentals
- The programming language doesn't matter!

Skill #2: Algorithmic Thinking (30%)

- **Algorithmic** (engineering, mathematical) **thinking**
  - The ability to analyze problems and find solutions
  - Breaking the problem down to steps (algorithm)
- How to develop algorithmic thinking?
  - Solve **1000+** programming **problems**
  - It takes 6 to 12 months of coding every day
- Courses in SoftUni: Programming Basics, Fundamentals and Advanced Modules
- The programming language doesn't matter!

Skill #3: Fundamental Concepts (25%)



- Fundamental software development concepts
  - **Object-oriented** programming (OOP)
  - **Functional** programming (FP)
  - **Asynchronous** programming and parallel execution
  - **Databases**: relational DB, SQL, document DB, key-value model
  - **Web technologies**: HTTP, JS, DOM, AJAX, REST, …
  - **Software engineering**: source control, agile, …
- **SoftUni** Curriculum: Professional Modules
- The programming language doesn't matter!

Skill #4: Languages & Technologies (25%)

- **Programming language and technologies**
  - They only form **25% of the skills** of a programmer!
- The programming languages and technologies come always together (as a **technology stack**)!
- Example of skills required for a **Junior C#** / **.NET Developer**:
  - C# + .NET Core + Visual Studio + databases + SQL Server + SQL + EF + ASP.NET MVC + HTML + CSS + JS + AJAX + REST + JSON + OOP + FP + algorithmic thinking + Git + software engineering + English + teamwork skills
- Software technologies **change very fast**!
- **SoftUni** Curriculum: Professional Modules

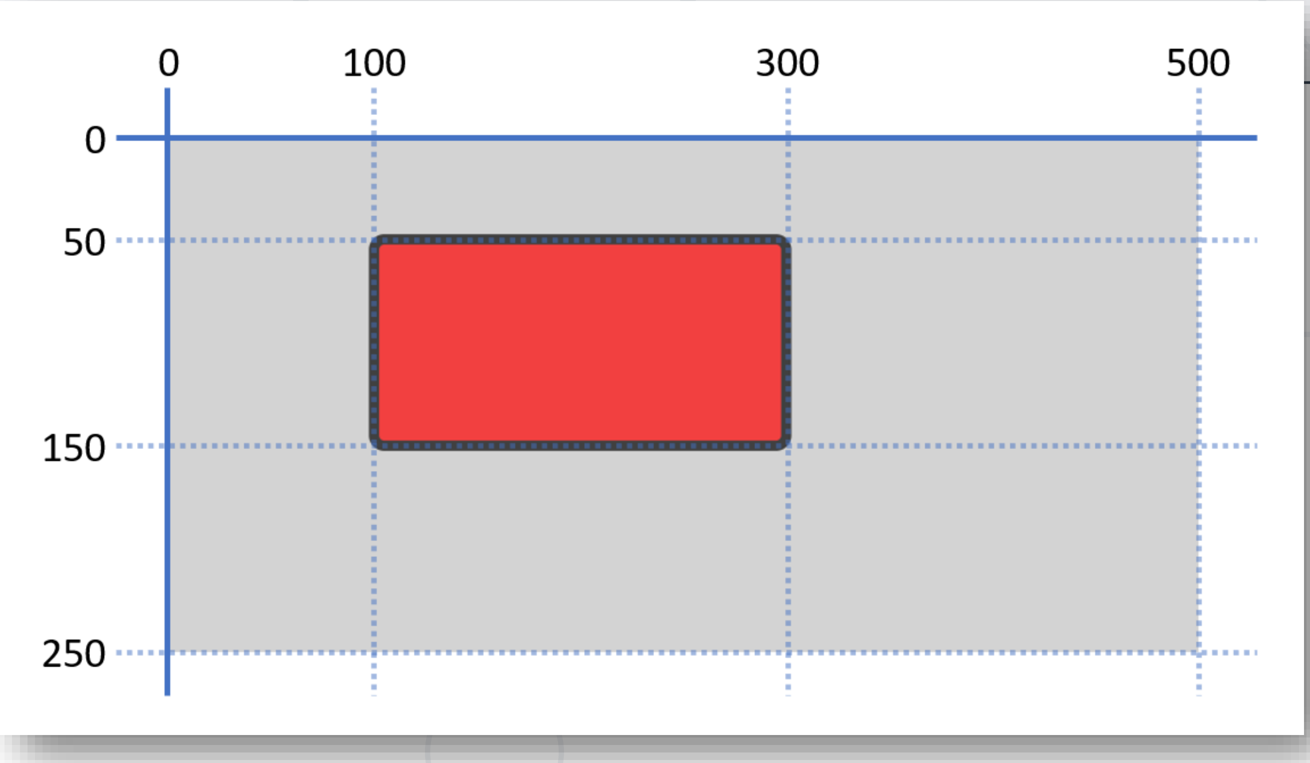
Fundamental Software Engineering Concepts

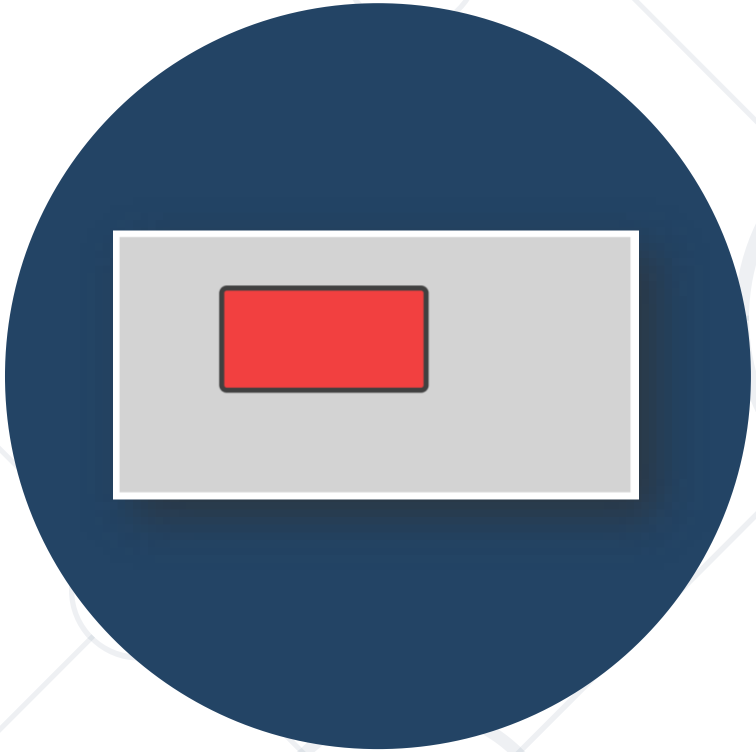
Math Concepts in Software Development

- Basic **mathematical concepts** related to programming
  - **Coordinate systems** (used in computer graphics)
  - Mathematical **functions** (lambda calculus, discrete functions, …)
  - **Vectors** and **matrices** (used in graphics, machine learning, …)
  - Finite state **automata** and **state machines** (used in parsers)
  - **Statistics** concepts (used in machine learning)
  - **Algorithm complexity** (estimate the speed)
  - Mathematical modeling

Coordinate System and SVG – Example

```
<svg width="500" height="250" style="background:lightgray">
  <rect x="100" y="50" width="200" height="100" rx="5" ry="5"
style="fill:red;stroke:black;stroke-width:5;opacity:0.7" />
</svg>
```





SVG and the Coordinate System

Live Demo

https://repl.it/@nakov/SVG-example

Object-Oriented Programming (OOP)

- **Object-Oriented Programming** (OOP) is the concept of using **classes** and **objects** (class instances) to model the real world

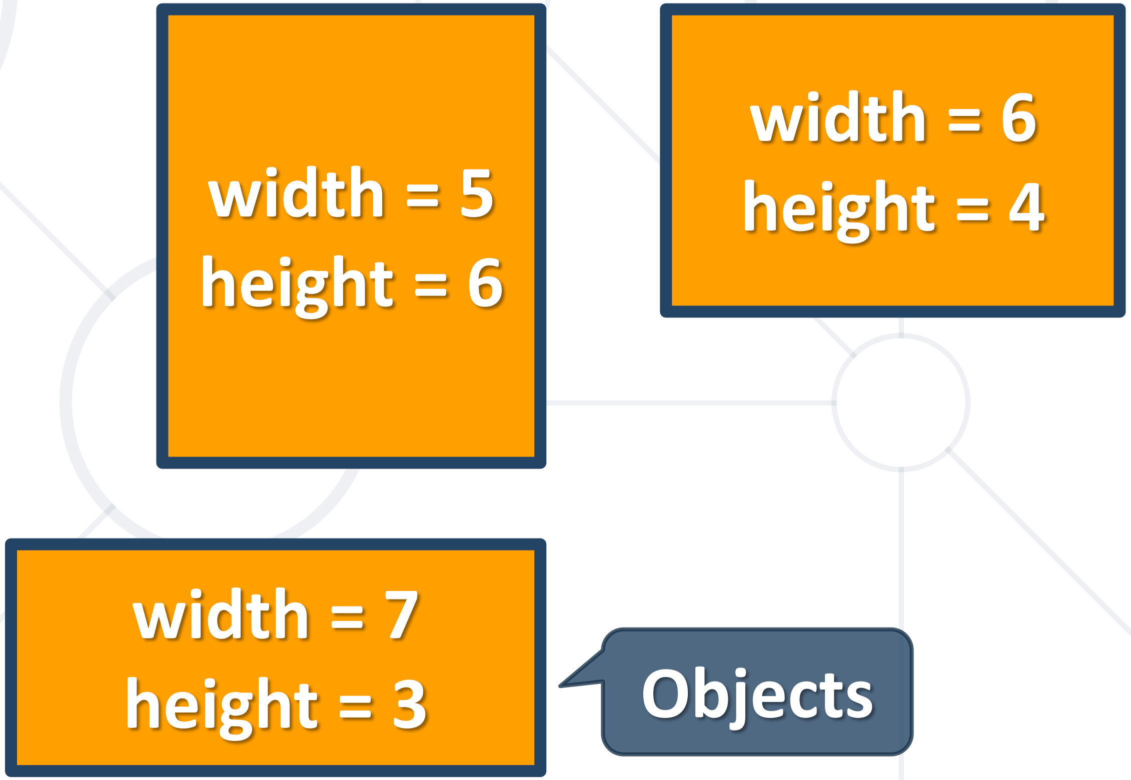
```
class Rectangle
{
  int width;
  int height;

  int CalcArea()
  {
    return width * height;
  }}
```

Class definition

Fields (data)

Methods (actions)



width = 5
height = 6

width = 6
height = 4

width = 7
height = 3

Objects

Object-Oriented Programming (OOP)

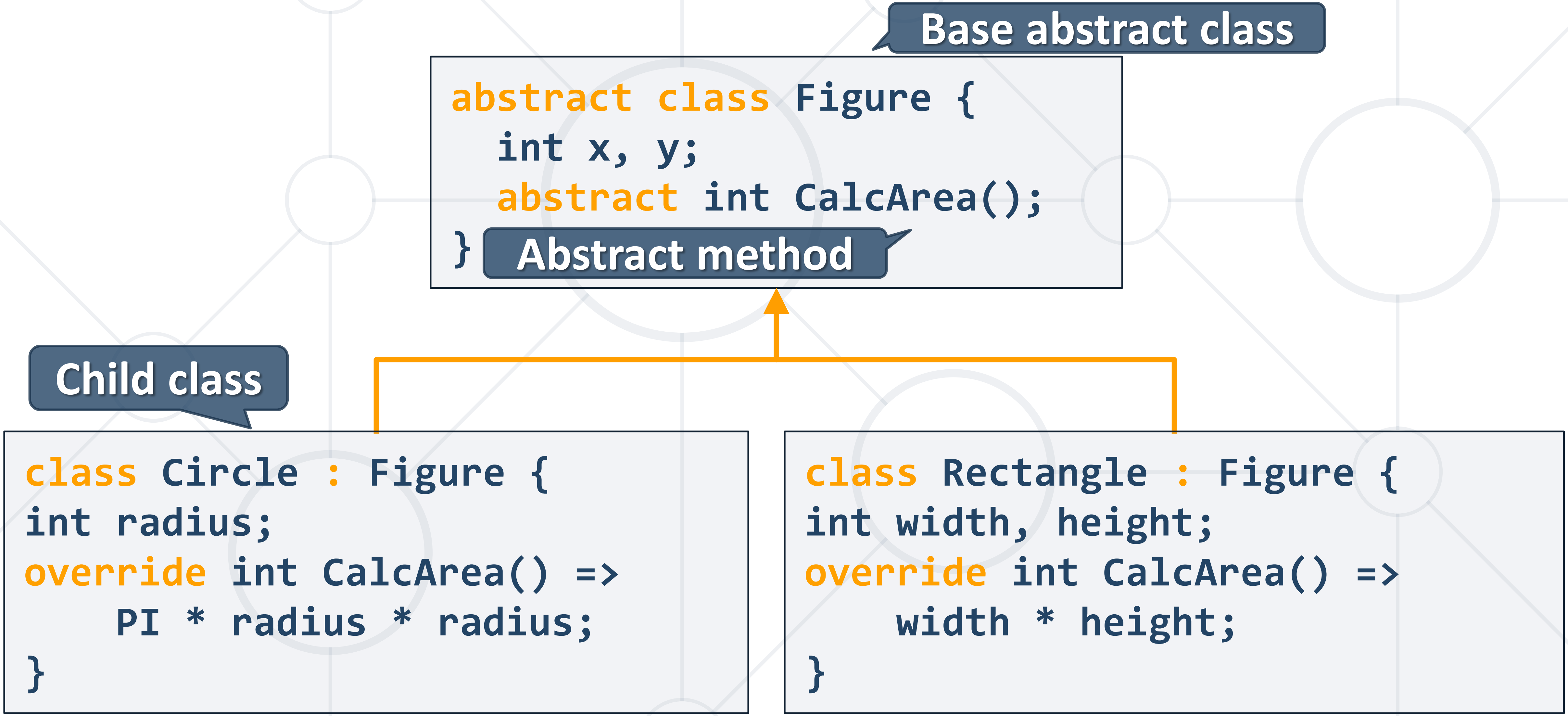
Live Demo

https://repl.it/@nakov/rectangle-oop-js
https://repl.it/@nakov/rectangle-oop-cs

Inheritance and Interfaces

- **Inheritance** allows classes to **inherit data and functionality** from a **parent class** (base class)
  - **Interface** – defines abstract actions
    - Actions to be implemented in descendent classes
  - **Abstract class** – abstraction, e.g. **`Figure`**
    - Defines data + actions + abstract actions
  - **Concrete class** – e.g. **`Circle`**, **`Rectangle`**
    - Defines data + concrete functionality

Inheritance and Interfaces – Example



Base abstract class

```
abstract class Figure {
  int x, y;
  abstract int CalcArea();
}
```

Abstract method

Child class

```
class Circle : Figure {
int radius;
override int CalcArea() =>
    PI * radius * radius;
}
```

```
class Rectangle : Figure {
int width, height;
override int CalcArea() =>
    width * height;
}
```

Inheritance in OOP

Live Demo

https://repl.it/@nakov/inheritance-oop-js

https://repl.it/@nakov/inheritance-oop-cs

Functional Programming

- **Functional programming** (FP)
  - Programming by composing **pure functions**, avoiding **shared state**, **mutable data**, and **side-effects**
  - **Declarative** programing approach (not **imperative**)
    - Program state flows through pure functions
- **Pure function** == function, which returns value only determined by its input, without side effects
  - Examples: *sqrt*(x), *sort*(list) → sorted list
  - Pure function == consistent result

Functional Programming Languages

- **Purely functional languages** are **unpractical** and rarely used
  - The program is **pure function** without side effects, e.g. **Haskell**
- **Impure functional languages**
  - Emphasize functional style, but allow side effects, e.g. **Clojure**
- **Multi-paradigm languages**
  - Combine multiple programing paradigms: **functional**, **structured**, **object-oriented**, …
  - Examples: **JavaScript**, **C#**, **Python**

Functional Programming – Examples

- Read several numbers and **find the biggest** of them (in C#)
  - **Functional** style

```
Console.WriteLine(
  Console.ReadLine()
    .Split(" ")
    .Select(int.Parse)
    .Max()
);
```

  - **Imperative** style

```
var input = Console.ReadLine();
var items = input.Split(" ");
var nums = items.Select(int.Parse);
var maxNum = nums.Max();
Console.WriteLine(maxNum);
```

Functional Programming (FP)

Live Demo

https://repl.it/@nakov/functional-max-num-cs

https://repl.it/@nakov/imperative-max-num-cs

Lambda and First-Class Functions

- **Lambda functions**: anonymous function (formula)

```
x => 2 * x
```

C#

```
x => 2 * x
```

JS

```
lambda x: 2 * x
```

Python

- JS, Python and C# and support **first-class functions** (functions can be stored in variables and passed as arguments)

```
let twice = x => 2 * x;
let d = twice(5);  // 10
```

JS

```
twice = lambda x: 2 * x
d = twice(5)  # 10
```

Python

```
Func<int, int> twice =
  x => 2 * x;
var d = twice(5);  // 10
```

C#

First-Class Functions

Live Demo

https://repl.it/@nakov/first-class-function-js

Higher Order Functions – Examples

- **Higher-order functions** take other functions as arguments

```
function aggregate(start, end, func) {
  for (var result = start, i = start+1; i <= end; i++)
    result = func(result, i);
  return result;
}
```

```
aggregate(1, 10, (a, b) => a + b)  // 55
```

```
aggregate(1, 10, (a, b) => a * b)  // 3628800
```

```
aggregate(1, 10, (a, b) => '' + a + b)  // "12345678910"
```

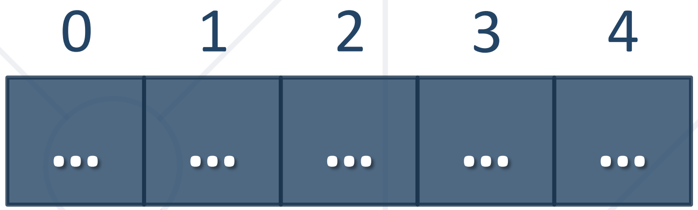
Higher-Order Functions

Live Demo

https://repl.it/@nakov/higher-order-functions-js

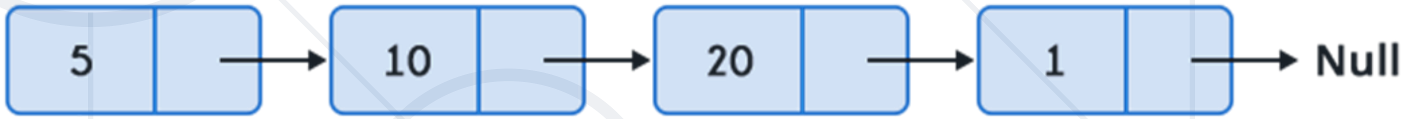
Data Structures

- **Data structures** are representations of data in the computer memory, which allow efficient access and modification
- **Linear data types**: arrays, lists, stacks, queues



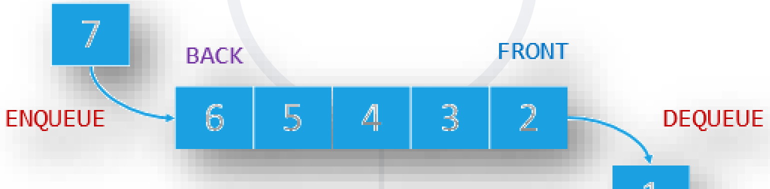
Array / list

(indexed group of elements)



Linked list

(sequence of linked elements)



Queue

List of Numbers – Example

- **List of numbers**, representing a sequence of income amounts

```
var incomes = [
  150, 200, 70.50, 120
];
```

Element	Value
incomes[0]	150
incomes[1]	200
incomes[2]	70.50
incomes[3]	120
incomes[4]	300

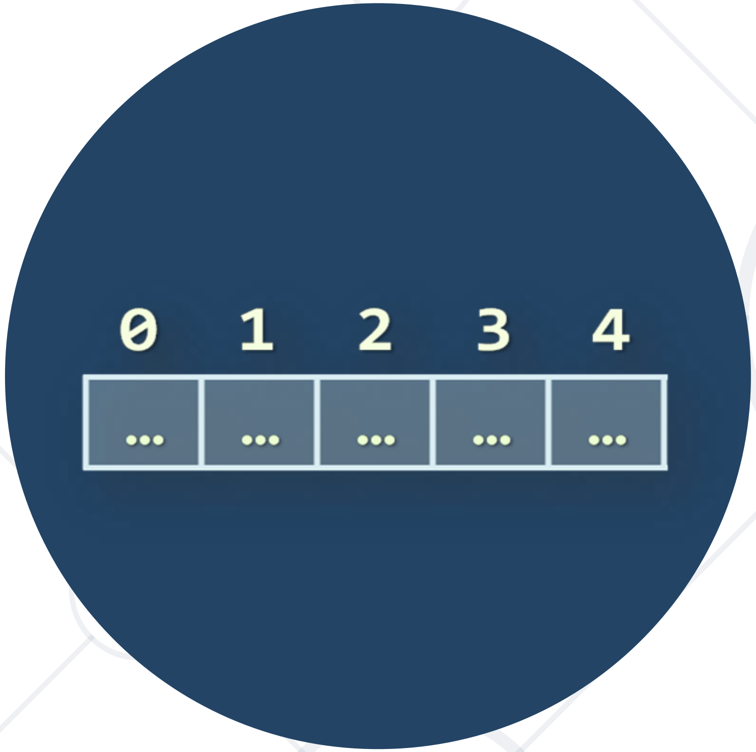
250

- **Adding** a new income

```
incomes.push(300);
```

- **Modifying** an existing income

```
incomes[1] = 250;
```



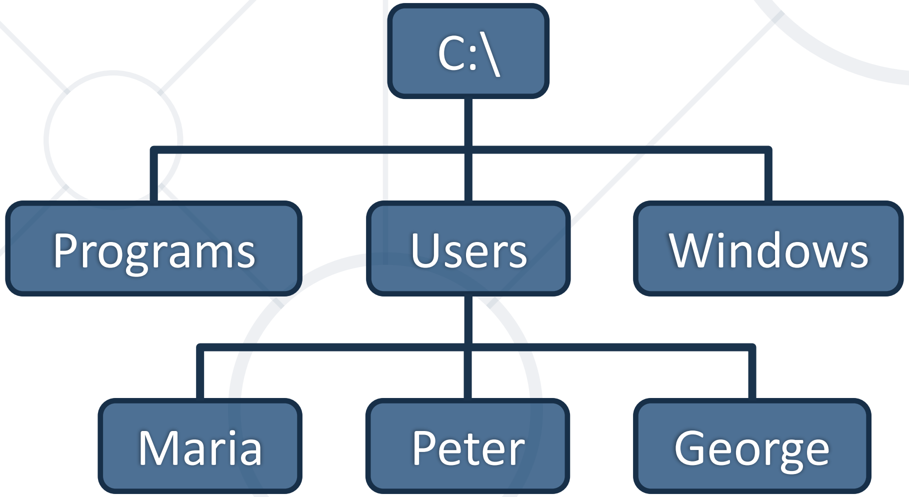
List of Numbers

Live Demo

https://repl.it/@nakov/list-example-js

Data Structures and Algorithms

- **Trees** and tree-like data structures
  - Each **node** holds data + list of **child nodes** + **parent node**

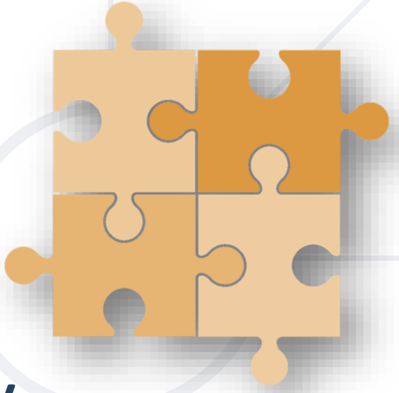


- **Tree traversal algorithms**
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)

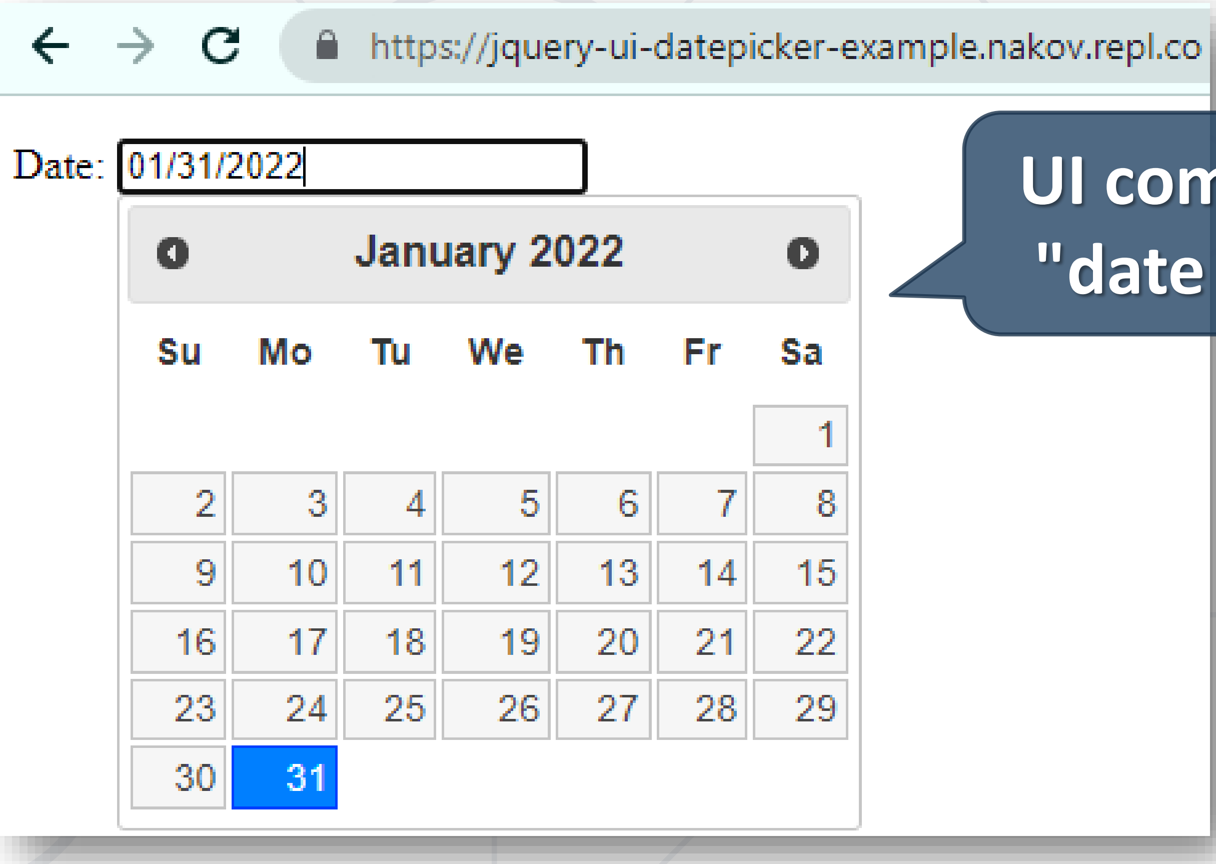
```
DepthFirstSearch(node) {
  print(node);
  for each ch in node.childNodes
    DepthFirstSearch(ch)
}
```

Component-Based Software Development

- **Component-based software development**
  - A **programming paradigm** in which applications are composed of re-usable **components**
- **Components** are self-contained pieces of functionality
  - e.g., PDF generator, email sender, date picker UI control
- User interface (UI) components are also known as **UI controls**, **visual components** or **widgets**
- Components are distributed in **libraries**
  - e.g., the UI control library jQuery UI



Example of Software Component



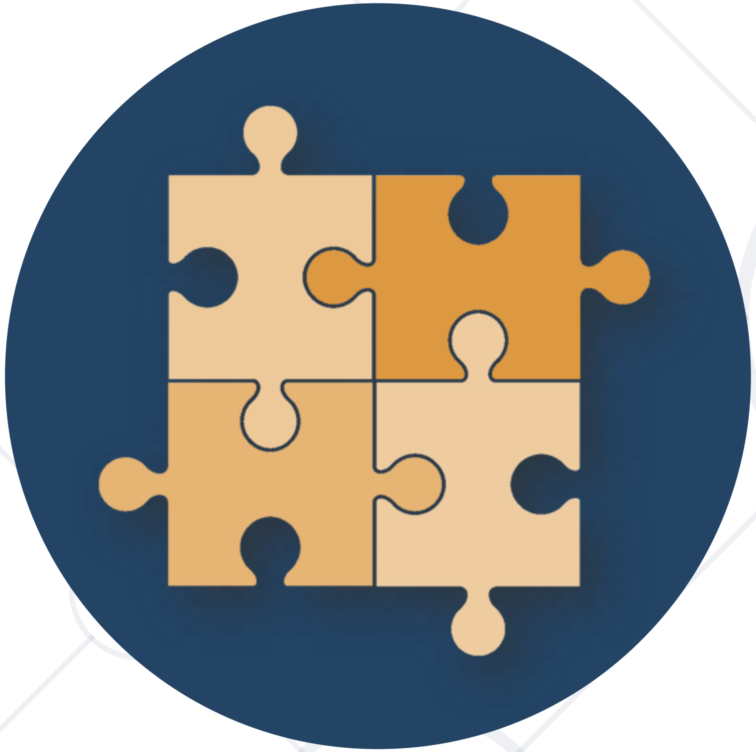
https://jquery-ui-datepicker-example.nakov.repl.co

Date: 01/31/2022

January 2022

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

UI component "date picker"



jQuery UI Date Picker

Live Demo

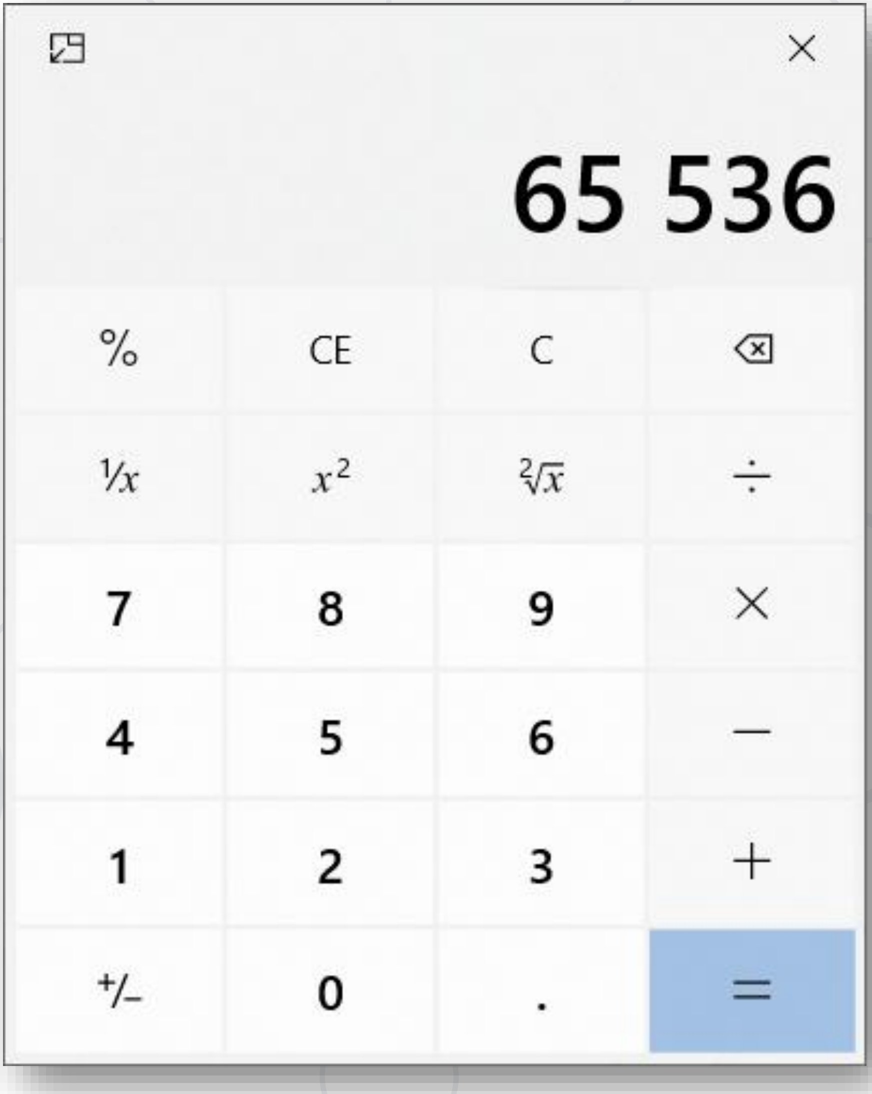
https://repl.it/@nakov/jquery-ui-datepicker-example

Event-Driven Programming

- **Event-driven programming**
  - A **programming paradigm** in which the flow of the program is determined by **events**, e.g., mouse clicks, key presses, etc.
- Event **source** (event emitter)
  - Produces events, e.g., when the mouse is clicked
- Event **handler** (event consumer, callback)
  - Processes events, e.g., show a message

Example of Event-Driven Programming

The UI framework draws the UI and check for events in a loop (event loop)

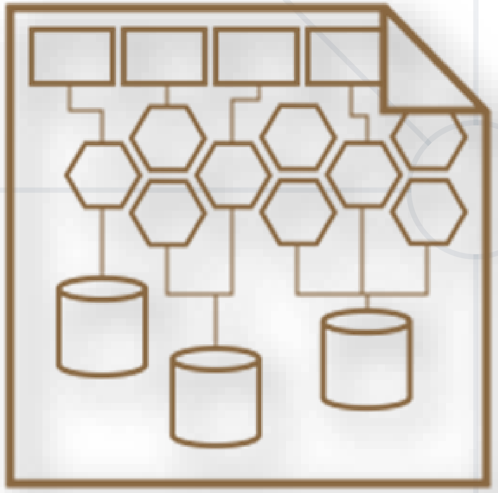


Clicking a button emits an event, which is handled by the calculator's engine

Software Architectures

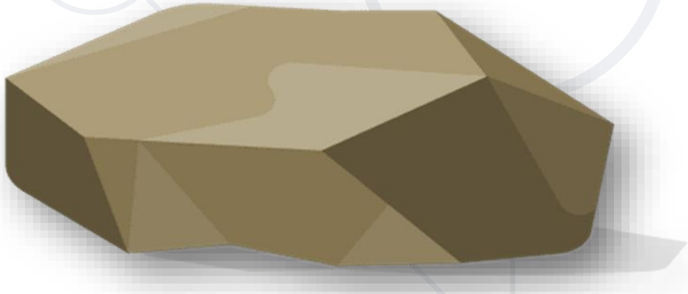
Software Architectures

- Software systems consist of **interconnected components** organized in certain structure called **architecture**
- Concepts related to **software architectures**
  - Monolith apps
  - Client-server model
  - Front-end and back-end
  - 3-tier and multi-tier architecture
  - SOA and microservices



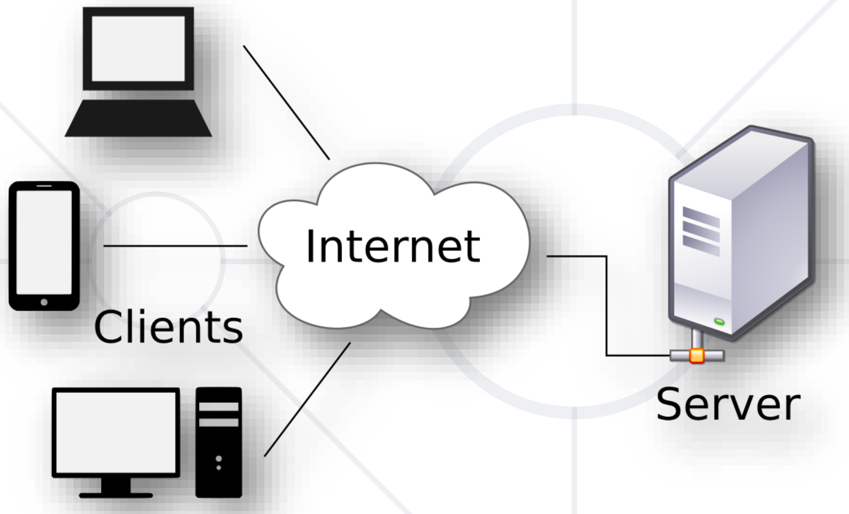
Monolith Apps

- **Monolith** apps
  - A **single application** holds its data, logic and user interface (UI)
  - **Single user** (no shared data access)
  - **Disconnected** from Internet
  - App data is stored on the **local machine**
  - Examples
    - A simple smartphone **game**
    - The **Notepad** text editor



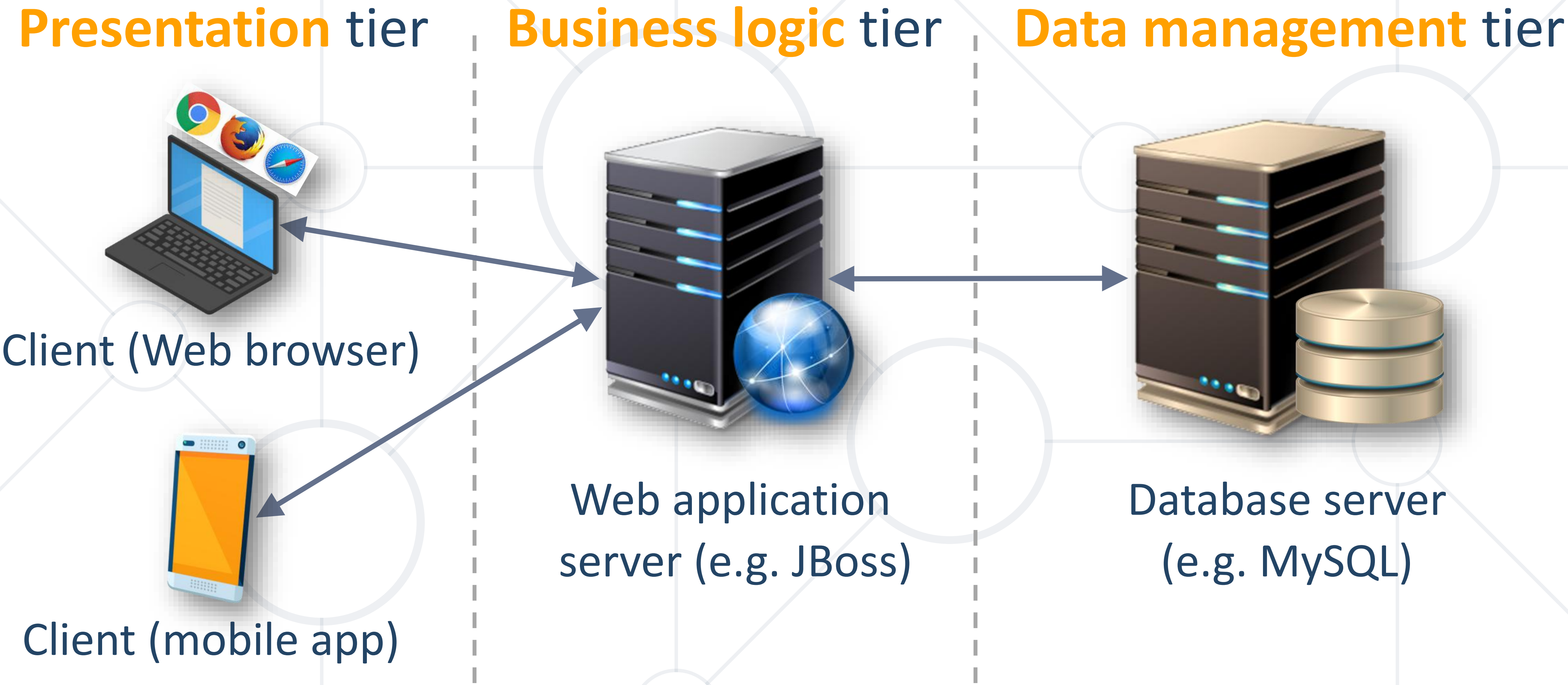
The "Client-Server" Model

- The **client-server** architectural model
  - The **server** holds app data and logic and provides APIs to clients
  - The **clients** implement the UI (the **user interface**) and consume the server APIs



- Examples
  - Web browser ↔ Web site
  - Email client ↔ Email server
  - Chat client ↔ Chat server

3-Tier Architecture / Multi Tier Architecture



Presentation tier

Business logic tier

Data management tier

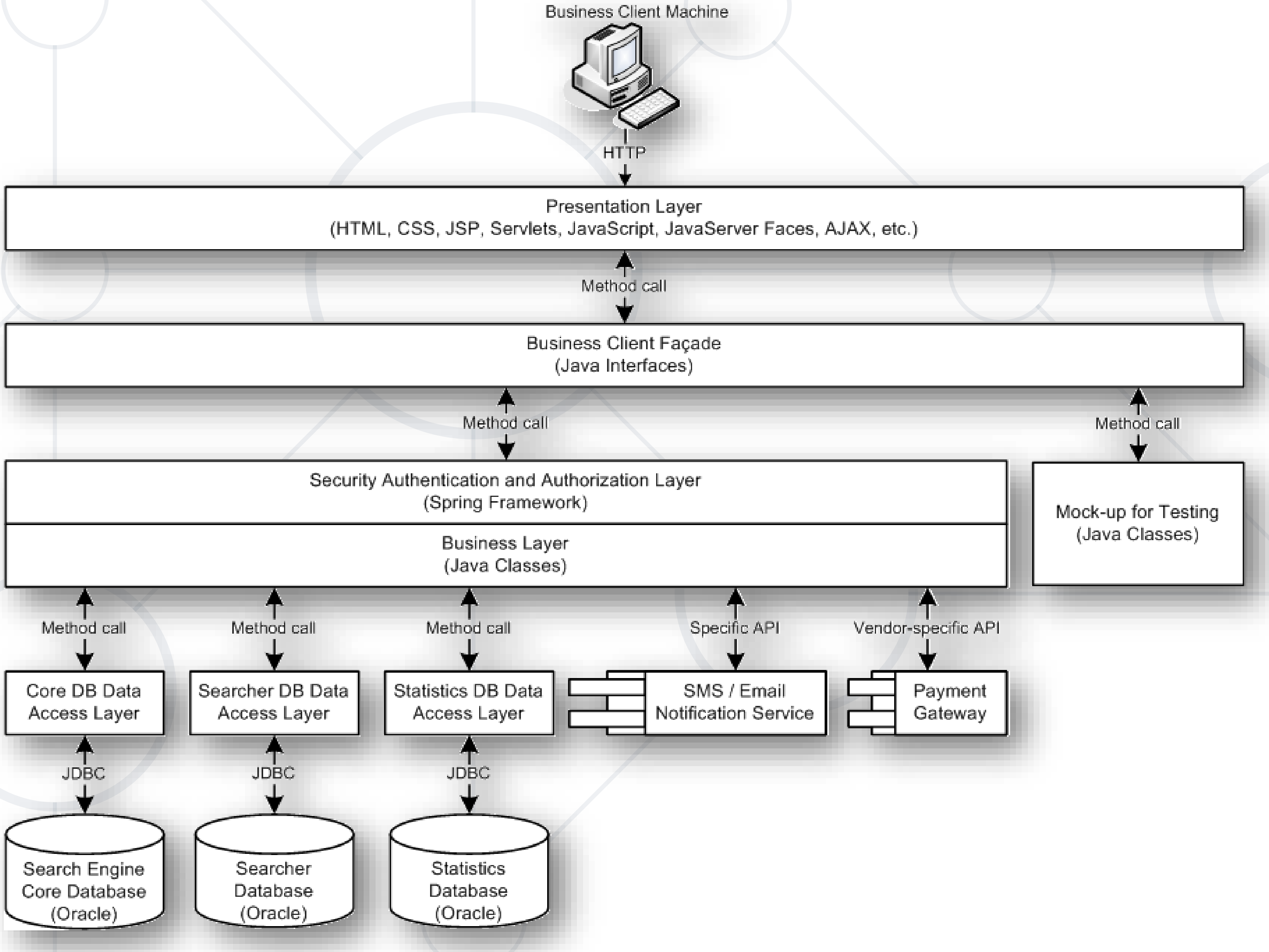
Client (Web browser)

Client (mobile app)

Web application server (e.g. JBoss)

Database server (e.g. MySQL)

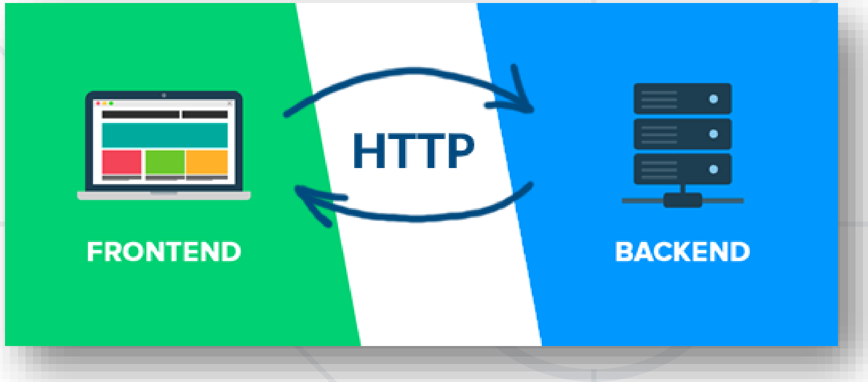
Software Architecture – Example



Business Client Machine
HTTP
Presentation Layer
(HTML, CSS, JSP, Servlets, JavaScript, JavaServer Faces, AJAX, etc.)
Method call
Business Client Façade
(Java Interfaces)
Method call
Method call
Security Authentication and Authorization Layer
(Spring Framework)
Business Layer
(Java Classes)
Mock-up for Testing
(Java Classes)
Method call
Method call
Method call
Specific API
Vendor-specific API
Core DB Data Access Layer
Searcher DB Data Access Layer
Statistics DB Data Access Layer
SMS / Email Notification Service
Payment Gateway
JDBC
JDBC
JDBC
Search Engine Core Database (Oracle)
Searcher Database (Oracle)
Statistics Database (Oracle)

Front-End and Back-End

- **Front-end** and **back-end** separate the modern apps into **client-side** (UI) and **server-side** (data) components
- **Front-end** == client-side components (presentation layer)
  - Implement the **user interface** (UI)
- **Back-end** == server-side components (data and business logic APIs)
  - Implements **data storage and processing**



- **HTTP** connects front-end with back-end

Front-End Technologies

- Front-end **technologies**
  - **Web front-end**: HTML + CSS + JavaScript + JS libraries
  - **Web front-end frameworks**: React, Angular, Vue, Flutter
  - **Desktop front-end**: XAML (Microsoft), UIKit (Apple)
  - **Mobile front-end**: Android UI, SwiftUI
  - **Hybrid mobile front-end**: React Native, Ionic
- **Front-end developers** deal with UI, UX and front-end technologies and frameworks

Back-End Technologies

- Back-end **technologies**: server-side frameworks and libraries
  - **C# / .NET back-end**: ASP.NET MVC, Web API, Entity Framework, …
  - **JavaScript back-end**: Node.js, Express.js / Meteor, MongoDB, …
  - **Python back-end**: Django / Flask, Django ORM / SQLAlchemy, …
  - **Java back-end**: Java EE, Spring MVC, Spring Data, Hibernate, …
  - **PHP back-end**: Apache, Laravel / Symfony, …
- **Back-end developers** deal with the business logic, data processing, data storage, APIs

Full Stack Development

- Full stack development
  - Combines **back end + front-end**
  - Requires end-to-end **architecture**, **design** and **implementation**
- **Full stack developers**
  - Build **back-end services**: business logic, data processing, data storage, databases, server-side APIs, containers and cloud
  - Build **front-end apps**: Web, mobile and desktop UI
  - Connect and **integrate** the **front-end** with the **back-end**

Summary

- The **4 skills** of the software engineers:
  - Coding
  - Problem Solving
  - Development Concepts
  - Software Technologies
- OOP, FP, Async, Event-Driven Programming
- Software Architectures
  - Front-End and Back-End