BASIC JAVA PROGRAMMING STUDY GUIDE

Produced by

Savvycle Online Tutoring

Content Area 1

Basics of Java

Key Concepts

- Java Syntax and Structure
  - Understand the basic structure of a Java program including classes, methods, and the main method.
- Variables, data types, and type conversion
  - Learn how to declare variables and work with different data types like integers, floats, strings, and booleans.
- Input and Output using System.out.println() and Scanner
  - Understand how to display output to the screen using `System.out.println()` and capture user input using the `Scanner` class.

Tips and Tricks:

- **Comment your code using `//` or `/* */`: to explain what each section does:**
  This will make your code more understandable for both yourself and others who read your code.

- **Use meaningful variable names to improve code readability:**
  Choose variable names that describe their purpose, making your code self-explanatory.

- **Experiment with Java IDEs like IntelliJ IDEA or Eclipse to test code snippets:**
  These IDEs provide features like code completion and debugging.

- **Start with simple programs to build a foundation before tackling complex projects:**
  Begin with basic exercises to grasp Java's fundamentals before moving on to more challenging tasks.

Essential Concepts:

- **Java is an object-oriented programming language known for its portability and robustness.**
- **Variables store data, and their types include int, double, String, and boolean.**
- **`System.out.println()` displays output to the screen, and `Scanner` captures user input.**
- **The `main` method is the entry point of any Java program, defined as `public static void main(String[] args)`. Every Java application must have a `main` method as the entry point.**
- **Classes define the structure of objects and contain methods and variables. Java programs are organized into classes. Each class is a blueprint for creating objects.**

Code-Based Examples:

TRY IT OUT: *Copy and Paste them in a .java file and try these out for yourself !*

1. Basic Class and Main Method:

```java
// Define a class named HelloWorld
public class HelloWorld {
    // The main method: entry point of the application
    public static void main(String[] args) {
        // Print a message to the console
        System.out.println("Hello, World!");
    }
}
```

2. Variable Declaration and Printing:

```
public class VariableExample {
    public static void main(String[] args) {
        // Declare a variable 'age' and assign your age to it
        int age = 25;

        // Print a message using the variable 'age'
        System.out.println("I am " + age + " years old.");
    }
}
```

3. Data Type Conversion:

```
public class TypeConversion {
    public static void main(String[] args) {
        // Convert a float to an integer
        float floatNum = 7.8f;
        int intNum = (int) floatNum;
        System.out.println(intNum);  // Output: 7

        // Convert an integer to a string
        int age = 25;
        String ageStr = Integer.toString(age);
        System.out.println("I am " + ageStr + " years old.");
    }
}
```

4. User Input:

```java
import java.util.Scanner;

public class UserInput {
    public static void main(String[] args) {
        // Create a Scanner object to read input
        Scanner scanner = new Scanner(System.in);

        // Ask the user for their name and print a personalized greeting
        System.out.print("What's your name? ");
        String name = scanner.nextLine();
        System.out.println("Hello, " + name + "! Nice to meet you.");
    }
}
```

Word Problem:

You are developing a simple weather app. The app should ask the user for the current temperature in Celsius and display the equivalent temperature in Fahrenheit.

The formula to convert Celsius to Fahrenheit is: **Fahrenheit = (Celsius * 9/5) + 32.**

Write a Java program that takes the temperature in Celsius as input and displays the converted temperature in Fahrenheit.

Answers:

Temperature Conversion:

```java
import java.util.Scanner;

public class TemperatureConverter {
    public static void main(String[] args) {
        // Create a Scanner object to read input
        Scanner scanner = new Scanner(System.in);

        // Ask the user to enter the temperature in Celsius
        System.out.print("Enter temperature in Celsius: ");
        double celsius = scanner.nextDouble();

        // Convert Celsius to Fahrenheit
        double fahrenheit = (celsius * 9/5) + 32;

        // Display the temperature in Fahrenheit
        System.out.println("Temperature in Fahrenheit: " + fahrenheit);
    }
}
```

Content Area 2

Control Structures and Loops

Key Concepts

- Conditional statements (`if`, `else if`, `else`)
  - Allow the program to make decisions based on conditions.
- Boolean expressions and logical operators for comparisons
  - Use expressions that evaluate to `true` or `false` to control program flow.

Logical Operators

- `&&` **(AND):** Both conditions must be true.
- `||` **(OR):** At least one condition must be true.
- `!` **(NOT):** Negates the condition
  - (if it's true, it becomes false and vice versa).

- Loops (`for`, `while`, `do-while`)
  - Enable repeated execution of a block of code.

Types of Loops

- `for` **Loop:** Best when you know the number of iterations.

```
// Example
for (int i = 0; i < 5; i++) {
    // Code to execute in each iteration
}
```

- **`while` Loop:** Best when the number of iterations is unknown.

```
while (condition) {
    // Code to execute as long as the condition is
true
}
```

- **`do-while` Loop:** Similar to **`while`,** but the code executes at least once.

```
do {
    // Code to execute
} while (condition);
```

Tips and Tricks:

- **Use Clear Conditions:**
  Ensure your if and loop conditions are clear and simple. Complex conditions can be broken into smaller checks..

- **Avoid Infinite Loops:**
  Always ensure that your loop has a condition that will eventually become false to avoid infinite loops.

- **Use Logical Operators Wisely:**
  Combine multiple conditions to make more complex checks.

- **Debugging Loops:**
  Use breakpoints or print statements to check if your loops behave as expected.

- **Write Readable Code:**
  Comment your code and use descriptive names for variables and methods to make your code easier to understand.

Essential Concepts:

- **Java File Structure:**
  - **main Method:** `The entry point of the application.`
  - **Class:** `A blueprint from which individual objects are created.`
  - **EXAMPLE**

```
public class Example {
    public static void main(String[] args) {
        // Your code here
    }
}
```

Code-Based Examples:

TRY IT OUT: *Copy and Paste them in a java file and try these out for yourself !*

1. Conditional Statements Example:

```
public class CheckAge {
    public static void main(String[] args) {
        int age = 20;
        if (age >= 18) {
            System.out.println("You are an adult.");
        } else {
            System.out.println("You are not an adult.");
        }
    }
}
```

2. for Loop Example:

```
public class ForLoopExample {
    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            System.out.println("This is iteration number " + (i + 1));
        }
    }
}
```

3. while Loop Example:

```
public class WhileLoopExample {
    public static void main(String[] args) {
        int count = 0;
        while (count < 3) {
            System.out.println("Count is " + count);
            count++;
        }
    }
}
```

4. do-while Example:

```
public class DoWhileExample {
    public static void main(String[] args) {
        int count = 0;
        do {
            System.out.println("Count is " + count);
            count++;
        } while (count < 3);
    }
}
```

5. Sum of Even Numbers Example:

```java
public class SumEvenNumbers {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 2; i <= 10; i += 2) {
            sum += i;
        }
        System.out.println("Sum of even numbers from 2 to 10 is " + sum);
    }
}
```

Word Problems:

1. Write a Java program to count from 1 to 10 using a for loop and print if the number is even or odd.

2. You are a teacher assigning grades to students based on their test scores. Write a program that takes a student's score as input and displays their corresponding grade. Use the following grading scale:
    - Score 90 or above: **A**
    - Score 80-89: **B**
    - Score 70-79: **C**
    - Score 60-69: **D**
    - Score below 60: **F**

Answers:

Even Odd Checker:

```
public class EvenOddCounter {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i % 2 == 0) {
                System.out.println(i + " is even.");
            } else {
                System.out.println(i + " is odd.");
            }
        }
    }
}
```

Grade Assignment:

```java
import java.util.Scanner;

public class GradeCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the student's test score: ");
        double score = scanner.nextDouble();

        String grade;
        if (score >= 90) {
            grade = "A";
        } else if (score >= 80) {
            grade = "B";
        } else if (score >= 70) {
            grade = "C";
        } else if (score >= 60) {
            grade = "D";
        } else {
            grade = "F";
        }

        System.out.println("Score: " + score);
        System.out.println("Grade: " + grade);
    }
}
```

Content Area 3

Arrays and Methods

Key Concepts

- ❖ Understand arrays as ordered collections of items
  - ➢ Arrays can store multiple values of the same type.
- ❖ Learn how to access, modify, and manipulate array elements
  - ➢ Use array indexing to work with individual elements.
- ❖ Define and use methods to encapsulate and reuse code
  - ➢ Methods are reusable blocks of code that perform specific tasks.

Tips and Tricks:

- **Arrays have a fixed size:**
  Arrays have a fixed size once declared. Use lists for dynamic collections.

- **Access elements in an array by index;**
  Access elements in an array using their index, starting from 0.

- **Utilize array methods like `length` to get the size of the array**

- **Use descriptive method names:**
  Use descriptive method names to clarify their purpose.

Essential Concepts:

- **Arrays are versatile data structures that can store multiple values.**
- **Methods help encapsulate and reuse code efficiently.**
- **Arrays and methods are fundamental for handling collections and structuring code.**

Code-Based Examples:

TRY IT OUT: *Copy and Paste them in a .java file and try these out for yourself !*

1. Array Declaration and Initialization:

```
public class ArrayExample {
    public static void main(String[] args) {
        // Declare and initialize an array of integers
        int[] numbers = {1, 2, 3, 4, 5};

        // Access and print each element in the array
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }
    }
}
```

2. Method Definition and Calling:

```
public class MethodExample {
    // Define a method to calculate the sum of two numbers
    public static int sum(int a, int b) {
        return a + b;
    }
    public static void main(String[] args) {
        // Call the sum method and print the result
        int result = sum(5, 7);
        System.out.println("Sum: " + result);
    }
}
```

3. Array Sum Method:

```
public class ArraySum {
    // Define a method to calculate the sum of elements in an array
    public static int arraySum(int[] array) {
        int sum = 0;
        for (int num : array) {
            sum += num;
        }
        return sum;
    }

    public static void main(String[] args) {
        // Declare and initialize an array of integers
        int[] numbers = {1, 2, 3, 4, 5};

        // Call the arraySum method and print the result
        int result = arraySum(numbers);
        System.out.println("Sum of array elements: " + result);
    }
}
```

Word Problem:

You are developing a library management system. Write a program that allows a librarian to store the titles of books in an array and find the total number of books in the library.

Answers:

Library Management System:

```
public class LibraryManagement {
    public static void main(String[] args) {
        // Declare and initialize an array of book titles
        String[] bookTitles = {"The Great Gatsby", "To Kill a Mockingbird",
"1984", "Moby Dick"};

        // Calculate and display the total number of books
        int totalBooks = bookTitles.length;
        System.out.println("Total number of books: " + totalBooks);

        // Display each book title
        for (String title : bookTitles) {
            System.out.println(title);
        }
    }
}
```

Content Area 4

Object-Oriented Programming (OOP)

Key Concepts

- Define classes and create objects
  - Classes are blueprints for creating objects with attributes and behaviors.
  - Objects are instances of classes.
- Understand constructors for initializing objects
  - Constructors are special methods used to initialize objects.
- Learn about access modifiers (`public`, `private`, `protected`)
  - Control the visibility and accessibility of class members.
- Implement inheritance to create hierarchical relationships between classes
  - Inheritance allows a class to inherit properties and methods from another class.
- Understand interfaces for defining methods that must be implemented by classes
  - Interfaces specify methods that classes must implement, ensuring a consistent API.

Tips and Tricks:

- **Use constructors:**
  Use constructors to initialize object attributes efficiently.

- **Apply access modifiers:**
  Apply access modifiers (i.e. getter and setter methods) to encapsulate class details and control access.

- **Utilize class inheritance:**
  Utilize inheritance to promote code reuse and establish class hierarchies.

- **Interfaces define contracts:**
  Interfaces define a contract for implementing classes, enhancing flexibility and consistency.

- **Organize classes into packages:**
  Organize related classes into packages for better project structure.

Essential Concepts:

- **Classes are blueprints for creating objects with attributes and behaviors.**
- **Constructors initialize object attributes.**
- **Access modifiers control visibility and accessibility.**
- **Inheritance allows classes to inherit properties and methods.**
- **Interfaces define methods that must be implemented by classes.**

Code-Based Examples:

TRY IT OUT: *Copy and Paste them in a .java file and try these out for yourself !*

1. Class and Object:

```java
public class Person {
    // Attributes of the class
    private String name;
    private int age;

    // Constructor to initialize attributes
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method to display person's details
    public void display() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }

    public static void main(String[] args) {
        // Create an object of the Person class
        Person person1 = new Person("Alice", 30);
        // Call the display method
        person1.display();
    }
}
```

2. Inheritance:

```java
// Base class
class Animal {
    // Method to display a message
    public void makeSound() {
        System.out.println("Animal sound");
    }
}

// Derived class inheriting from Animal
class Dog extends Animal {
    // Override the makeSound method
    @Override
    public void makeSound() {
        System.out.println("Bark");
    }
}

public class InheritanceExample {
    public static void main(String[] args) {
        // Create an object of the Dog class
        Dog dog = new Dog();
        // Call the makeSound method
        dog.makeSound();  // Output: Bark
    }
}
```

3. Interface Implementation:

```
// Define an interface
interface Printable {
    void print();
}

// Implement the Printable interface in a class
class Document implements Printable {
    private String content;

    // Constructor to initialize content
    public Document(String content) {
        this.content = content;
    }

    // Implement the print method
    @Override
    public void print() {
        System.out.println(content);
    }
}

public class InterfaceExample {
    public static void main(String[] args) {
        // Create an object of the Document class
        Document doc = new Document("Hello, Interface!");
        // Call the print method
        doc.print();
    }
}
```

Word Problem:

You are developing a banking application. Create classes for **Account** and **SavingsAccount** that inherit from **Account**. Implement methods to **deposit, withdraw, and display** the account balance. The **SavingsAccount** should have an additional method to calculate and add interest.

Answers:

Banking Application:

```
// Base class Account
class Account {
    private double balance;

    // Constructor to initialize balance
    public Account(double balance) {
        this.balance = balance;
    }

    // Method to deposit money
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: " + amount);
    }

    // Method to withdraw money
    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println("Insufficient balance");
        }
    }

    // Method to display balance
    public void displayBalance() {
        System.out.println("Balance: " + balance);
    }
}

// Derived class SavingsAccount
class SavingsAccount extends Account {
    private double interestRate;
```

```
    // Constructor to initialize balance and interest rate
    public SavingsAccount(double balance, double interestRate) {
        super(balance);
        this.interestRate = interestRate;
    }

    // Method to calculate and add interest
    public void addInterest() {
        double interest = (super.balance * interestRate) / 100;
        super.deposit(interest);
        System.out.println("Interest added: " + interest);
    }
}

public class BankingApp {
    public static void main(String[] args) {
        // Create an object of the SavingsAccount class
        SavingsAccount account = new SavingsAccount(1000, 5);

        // Deposit money
        account.deposit(200);

        // Withdraw money
        account.withdraw(150);

        // Add interest
        account.addInterest();

        // Display balance
        account.displayBalance();
    }
}
```

Content Area 5

Advanced Topics

Key Concepts

- Exception handling to manage runtime errors
  - Use `try, catch, finally` blocks to handle exceptions and ensure program stability.
- File I/O for reading from and writing to files
  - Utilize `FileReader, BufferedReader, FileWriter,` and `BufferedWriter` for file operations.
- Collections framework for advanced data structures
  - Explore `ArrayList`, `HashMap`, `HashSet`, and other collection classes for handling dynamic collections.
- Threads for concurrent programming
  - Understand how to create and manage threads for multitasking in Java..
- Networking basics for client-server communication:
  - Use `Socket` and `ServerSocket` classes for network programming.

Tips and Tricks:

- **Always handle exceptions:**
  Always handle exceptions to prevent your program from crashing unexpectedly.Lorem ipsum.

- **Use buffered streams:**
  Use buffered streams for efficient file I/O operations.

- **Leverage the Collections framework:**
  Leverage the Collections framework for flexible and powerful data management.

- **Implement concurrency:**
  Implement concurrency with threads to enhance performance in multitasking scenarios.

- **Explore network programming:**
  Explore network programming for client-server applications and real-time communication.

Essential Concepts:

- **Exception handling ensures program stability by managing runtime errors.**
- **File I/O operations facilitate reading from and writing to files.**
- **The Collections framework provides versatile data structures for advanced data management.**
- **Threads enable concurrent programming for multitasking.**
- **Networking basics are essential for client-server communication and real-time applications.**

Code-Based Examples:

TRY IT OUT: *Copy and Paste them in a java file and try these out for yourself !*

1. Exception Handling:

```
public class ExceptionHandling {
    public static void main(String[] args) {
        try {
            // Attempt to divide by zero
            int result = 10 / 0;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            // Handle the arithmetic exception
            System.out.println("Error: Division by zero is not allowed.");
        } finally {
            // Code that always executes
            System.out.println("Operation completed.");
        }
    }
}
```

2. File I/O:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class FileIOExample {
    public static void main(String[] args) {
        // Write to a file
        try (FileWriter writer = new FileWriter("example.txt")) {
            writer.write("Hello, File I/O!");
        } catch (IOException e) {
            System.out.println("Error writing to file: " + e.getMessage());
        }

        // Read from a file
        try (BufferedReader reader = new BufferedReader(new
FileReader("example.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading from file: " +
e.getMessage());
        }
    }
}
```

3. Collections Framework:

```java
import java.util.ArrayList;
import java.util.HashMap;

public class CollectionsExample {
    public static void main(String[] args) {
        // ArrayList example
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");

        System.out.println("Fruits: " + fruits);

        // HashMap example
        HashMap<String, Integer> scores = new HashMap<>();
        scores.put("Alice", 85);
        scores.put("Bob", 92);
        scores.put("Charlie", 78);

        System.out.println("Scores: " + scores);
    }
}
```

4. Threads:

```
public class ThreadExample extends Thread {
    // Define the run method for the thread
    public void run() {
        System.out.println("Thread is running");
    }

    public static void main(String[] args) {
        // Create and start a new thread
        ThreadExample thread = new ThreadExample();
        thread.start();
    }
}
```

5. Networking:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class NetworkingExample {
    public static void main(String[] args) {
        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server is listening on port 12345");

            // Accept a client connection
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected");

            // Read from client
            BufferedReader input = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            String clientMessage = input.readLine();
            System.out.println("Client says: " + clientMessage);

            // Send response to client
            PrintWriter output = new PrintWriter(new
OutputStreamWriter(clientSocket.getOutputStream()), true);
            output.println("Hello, Client!");

            // Close connections
            input.close();
            output.close();
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Word Problem:

You are developing a chat application that allows multiple clients to connect to a server and exchange messages. Write a program to create a simple server that listens for client connections and responds with a greeting message.

Answers:

Chat Server:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class ChatServer {
    public static void main(String[] args) {
        try {
            // Create a server socket
           ServerSocket serverSocket = new ServerSocket(12345);
           System.out.println("Chat server is listening on port 12345");

            while (true) {
                // Accept a client connection
               Socket clientSocket = serverSocket.accept();
               System.out.println("Client connected");

                // Create a new thread for each client
                new Thread(() -> {
                    try {
                        // Read from client
                      BufferedReader input = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                       String clientMessage;
                       while ((clientMessage = input.readLine()) != null) {
                           System.out.println("Client says: " + clientMessage);

                           // Send response to client
                           PrintWriter output = new PrintWriter(new
OutputStreamWriter(clientSocket.getOutputStream()), true);
                           output.println("Server received: " + clientMessage);
                       }


                       // Close connections
```

```
                    input.close();
                clientSocket.close();
            } catch (IOException e) {
                System.out.println("Error: " + e.getMessage());
            }
        }).start();
    }
} catch (IOException e) {
    System.out.println("Error: " + e.getMessage());
}
}
```

Glossary

⇨ Content Area 1: Introduction to Java

Syntax: The set of rules that dictate how programs written in a programming language should be structured.

Variables: Named storage locations that hold data values.

Data Types: Categories that define the type of data a variable can hold, such as integers, floats, strings, and booleans.

Type Conversion: The process of changing the data type of a value.

`System.out.println():` A method in Java used to display output on the screen.

`Scanner:` A class in Java used to capture user input.

⇨ Content Area 2: Control Structures and Loops

Control Structures: Statements that determine the flow of execution in a program, including if, elif, and else statements.

Loops: Repeating a set of statements multiple times, such as the for loop and while loop.

Conditionals: Statements that perform actions based on certain conditions.

Iteration: The process of repeating a set of instructions in a loop.

Break Statement: Used to exit a loop prematurely.

⇨ Content Area 3: Arrays and Methods

Arrays: Ordered collections of items of the same type, indexed by integers.

Indexing: Accessing individual elements in an array using their positions.

Methods: Blocks of reusable code that perform a specific task.

Parameters: Values that are passed to a method when it's called.

Return Value: The value that a method generates and returns when it's executed.

Enhanced for Loop: A concise way to iterate over elements of an array or a collection.

⇨ Content Area 4: Collections Framework and File Handling

Collections Framework: A unified architecture for representing and manipulating collections, including classes like ArrayList, HashMap, and HashSet.

ArrayList: A resizable array implementation in the Java Collections Framework.

HashMap: A collection that stores key-value pairs.

File Handling: Reading from and writing to files on a computer.

BufferedReader: A class used to read the text from an input stream.

FileWriter: A class used to write characters to a file.

⇨ Content Area 5: Advanced Java Concepts

Object-Oriented Programming (OOP): A programming paradigm that uses objects and classes to organize code.

Class: A blueprint for creating objects, containing attributes (variables) and methods (functions).

Object: An instance of a class that contains data and behavior.

Inheritance: The ability of a class to inherit attributes and methods from another class.

Polymorphism: The ability of different classes to be treated as instances of the same class through inheritance.

Exception Handling: A mechanism to handle errors and prevent program crashes.

Additional Resources

Online Resources:

Oracle Java Documentation:

- **The official Java documentation offers tutorials, API documentation, and resources.**

Java Programming and Software Engineering Fundamentals:

- **A Coursera course that covers the basics and more advanced topics.**

GeeksforGeeks Java Tutorial:

- **In-depth tutorials and articles on various Java topics.**

Stack Overflow:

- **A community where you can ask questions and find answers related to Java programming.**

Replit:

- **An online IDE and platform for coding, sharing, and collaborating on Java projects.**

Tips and Resources:

Debugging Skills:

- **Teaches techniques for identifying and fixing errors in code.**

Online Coding Platforms:

- **You may practice on platforms like Codecademy, LeetCode, or HackerRank.**

Community and Forums:

- **Explore Java communities like Stack Overflow for problem-solving.**

Project-Based Learning:

- **Try building small projects to apply learned concepts practically.**

Version Control:

- **You may use Git for collaborative coding.** `(www.github.com)`

"Java is to JavaScript what car is to Carpet."

- Chris Heilmann, JavaScript Developer