INTRODUCTION TO JAVA

Java 1.0

JavaGuru

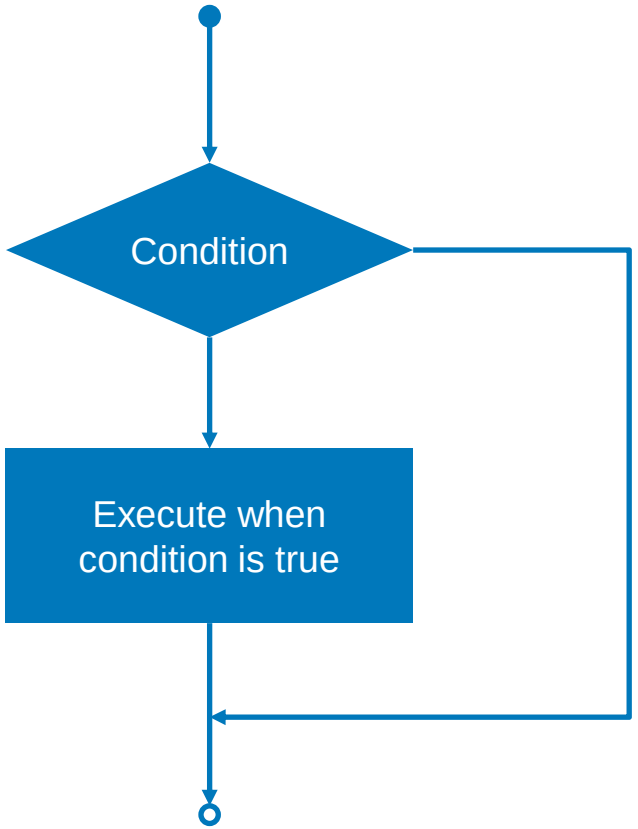
CONDITIONAL FLOW CONTROL

Lesson # 03

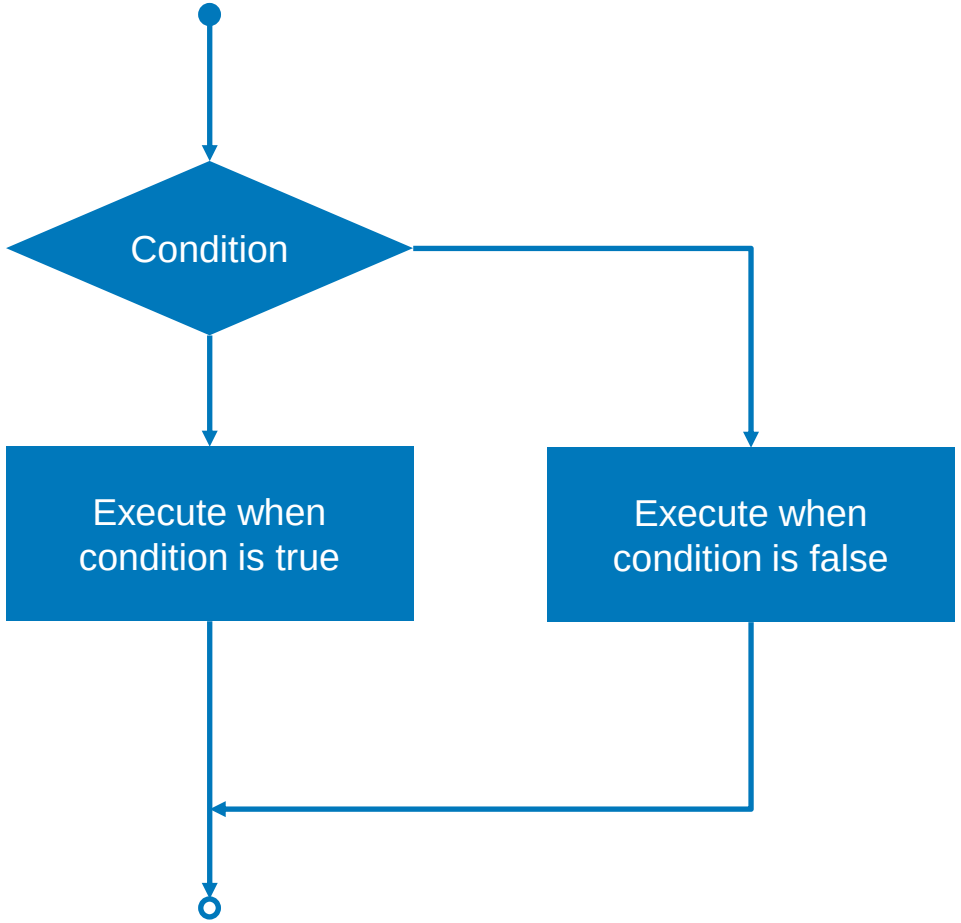
CONDITIONAL STATEMENTS

- Control code execution by specifying certain conditions
  - When conditional statement is met (equals to 'true')
  - When conditional statement is not met (equals to 'false')
- There are two main conditional statements:
  - If statement
  - Switch statement

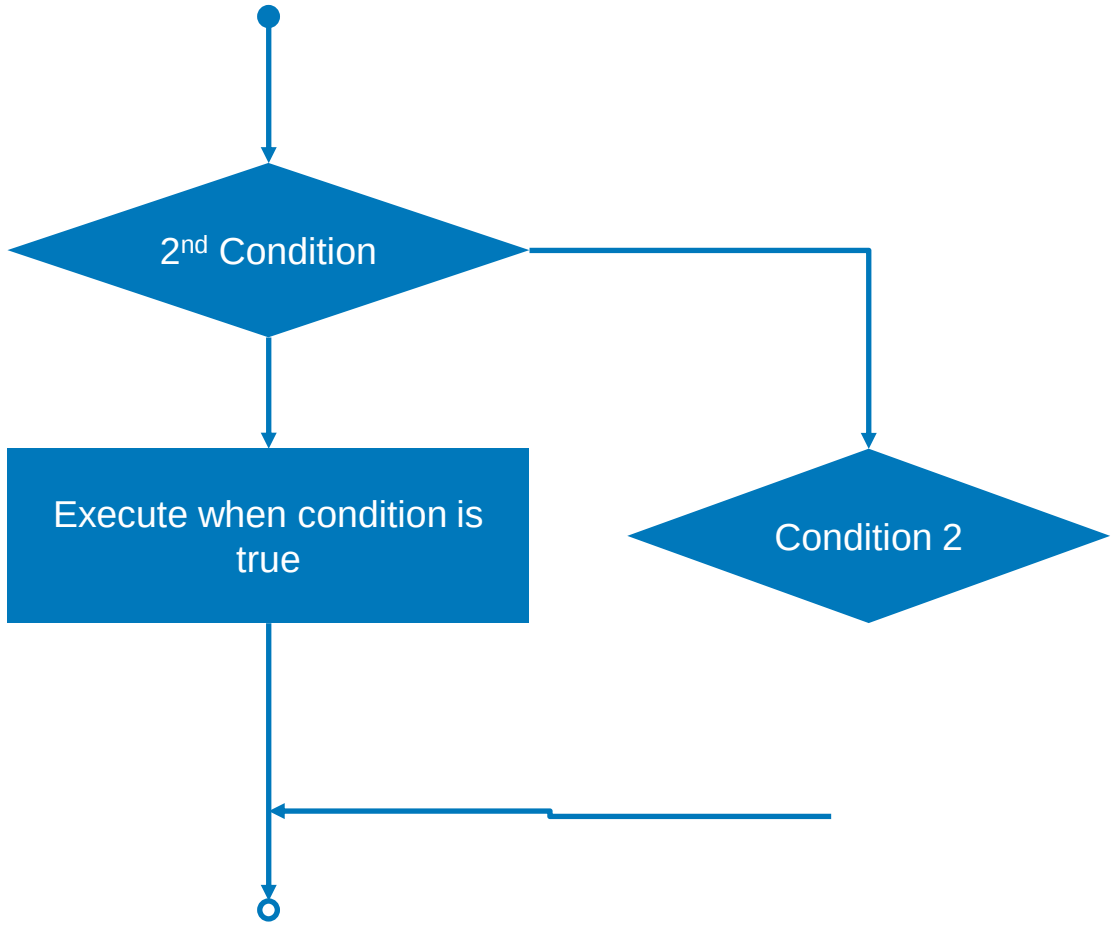
IF – DECISION MAKING FLOWCHART



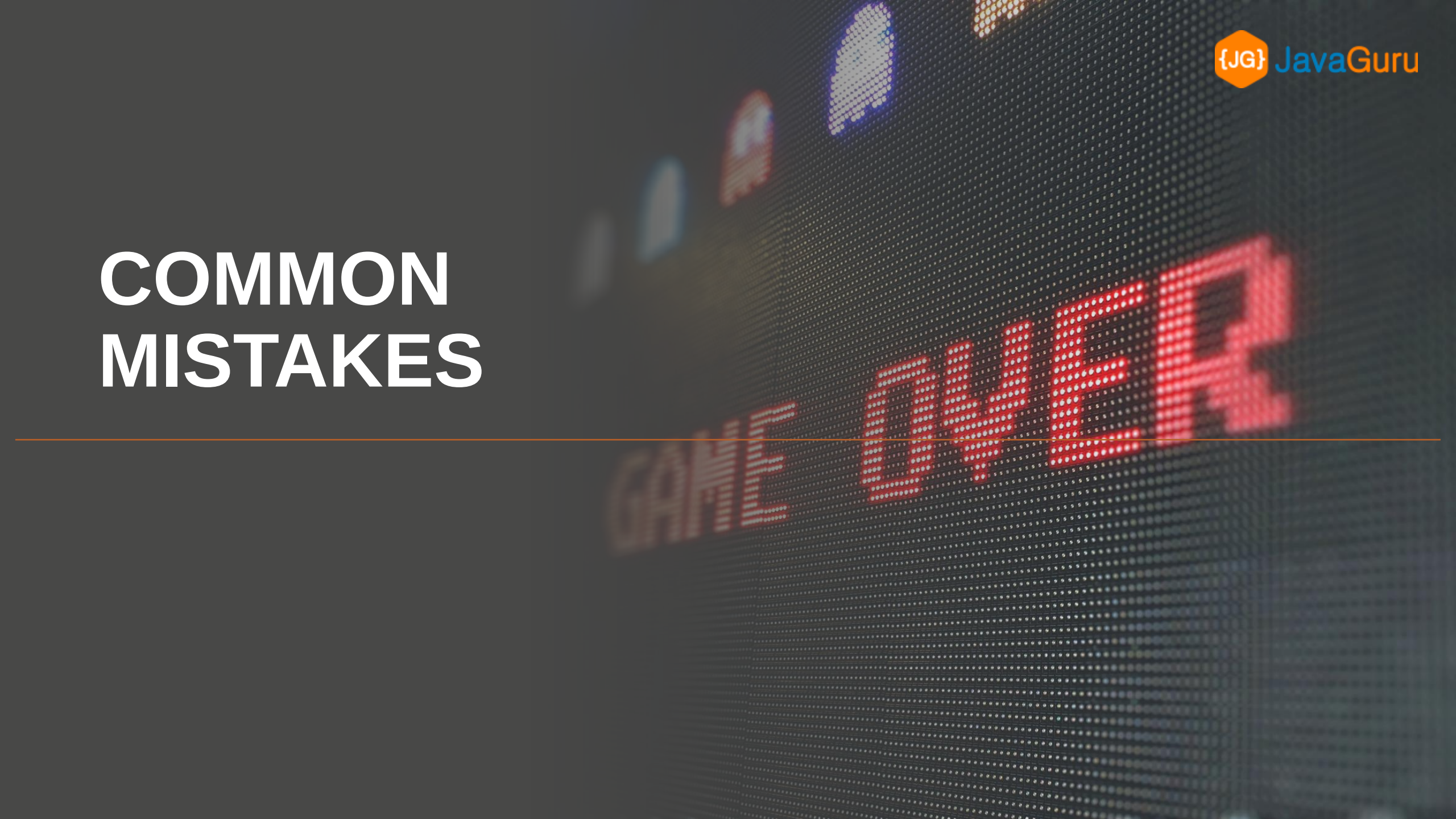
IF-ELSE - DECISION MAKING FLOWCHART



IF-ELSE-IF - DECISION MAKING FLOWCHART



COMMON MISTAKES



COMMON ERRORS

1. Missing terminator sign ';'
2. Incorrect spelling
    1. Class name
    2. Package name
    3. Variable name
3. Code placement outside of the body
4. Missing quotes or misplacement

MISSING TERMINATOR SIGN

```
public static void main(String[] args) {
    System.out.println("I forgot to add semicolon")
}
```

MISSING TERMINATOR SIGN

```
public static void main(String[] args) {
    System.out.println("I forgot to add semicolon");
}
```

BAD CLASS SPELLING

```
public class misspelledNames {

    public static void main(String[] args) {
        system.out.println("Oops, something went wrong");
    }

}
```

BAD CLASS SPELLING

```
public class MisspelledNames {

    public static void main(String[] args) {
        System.out.println("Oops, something went wrong");
    }

}
```

BAD PACKAGE SPELLING

```
package lv.javaguru.demo.HOMEwork;
```

BAD PACKAGE SPELLING

```
package lv.javaguru.demo.homework;
```

CODE PLACEMENT OUTSIDE OF THE BODY

```
public class MisplacedCode {

    System.out.println("Hello world!");
    public static void main(String[] args) {

    }

}
```

CODE PLACEMENT OUTSIDE OF THE BODY

```
public class MisplacedCode {


    public static void main(String[] args) {
        System.out.println("Hello world!");
    }

}
```

QUOTES MISPLACEMENT

```
public class CoffeeTime {

    public static void main(String[] args) {
        System.out.println(I want my coffe);
        System.out.println("It energises me);
    }

}
```

QUOTES MISPLACEMENT

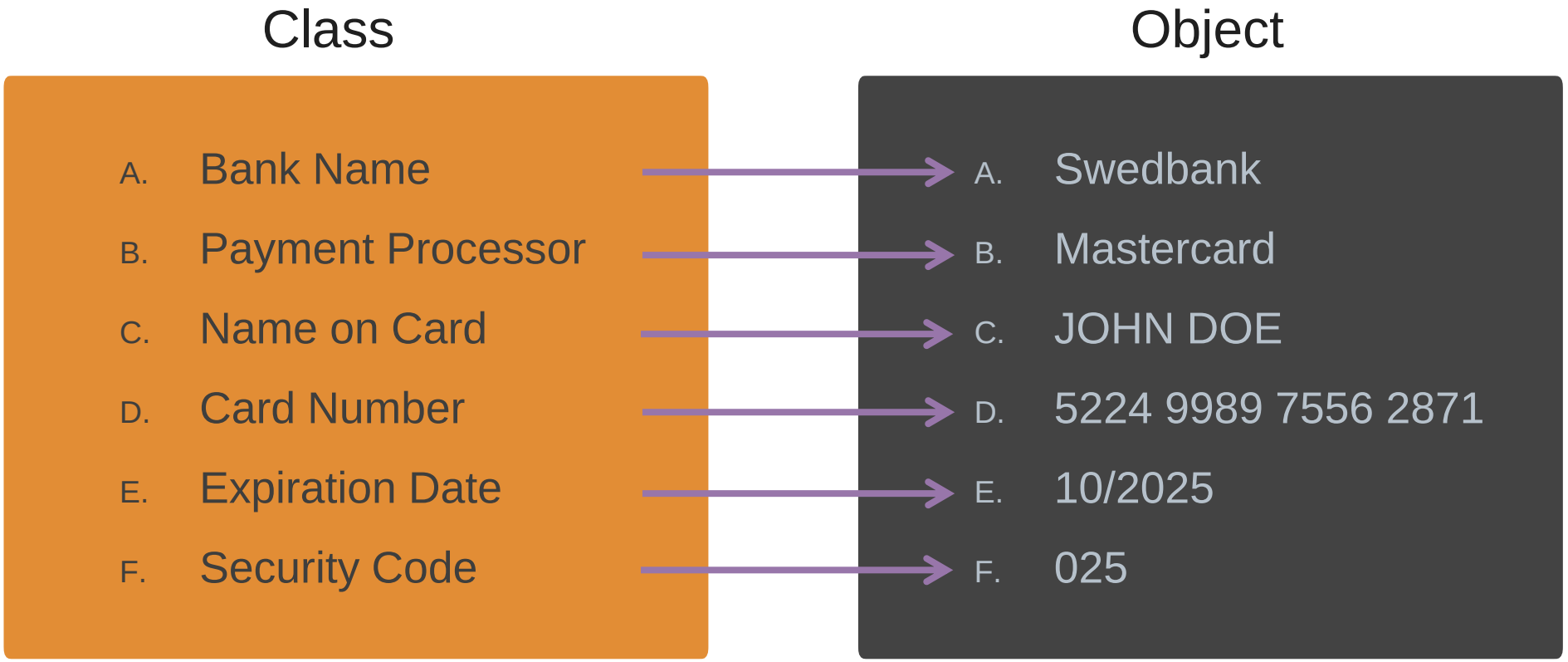
```
public class CoffeeTime {

    public static void main(String[] args) {
        System.out.println("I want my coffee");
        System.out.println("It energises me");
    }

}
```

OBJECT - ORIENTED PROGRAMMING

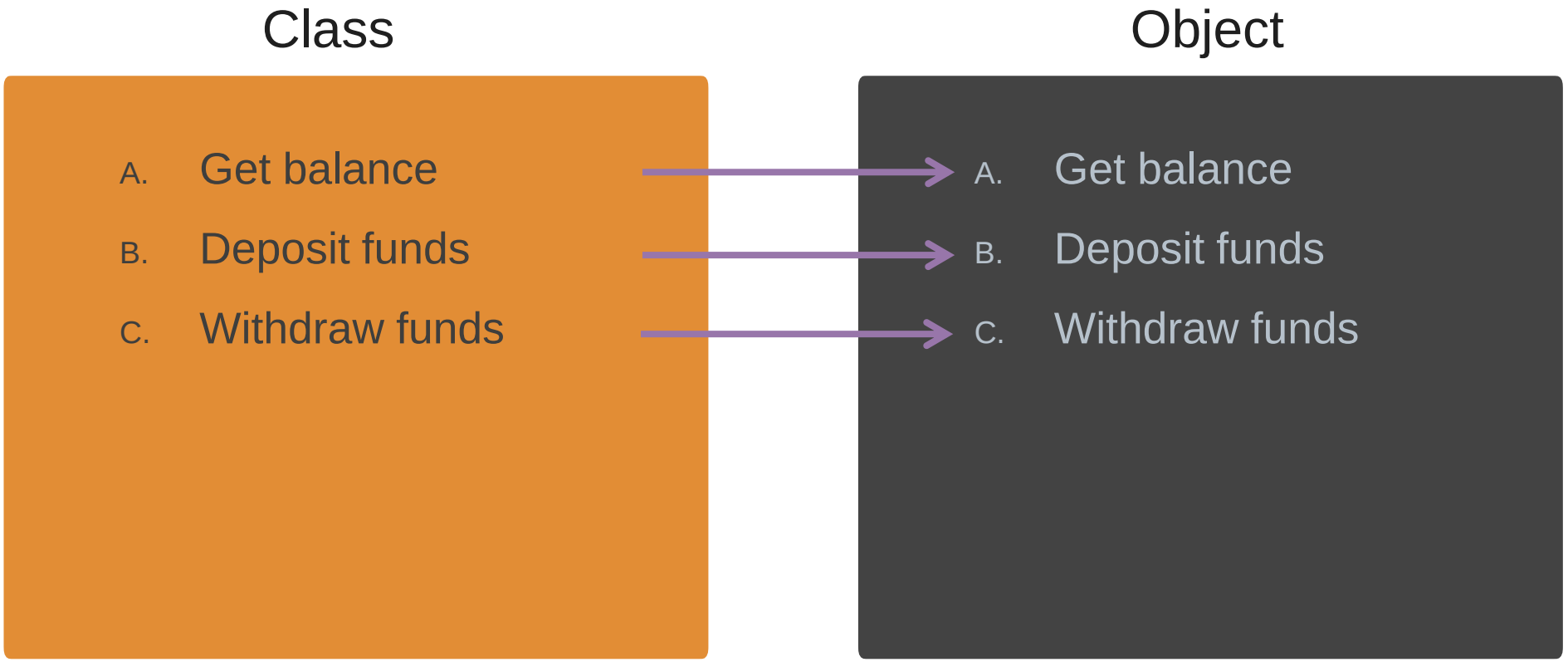
CONCEPTS

1. Class describes template (blueprint) of something with state and behavior
2. Object is concrete instance of that class with set state

EXAMPLE – BANK CARD (STATE)



EXAMPLE – BANK CARD (BEHAVIOR)



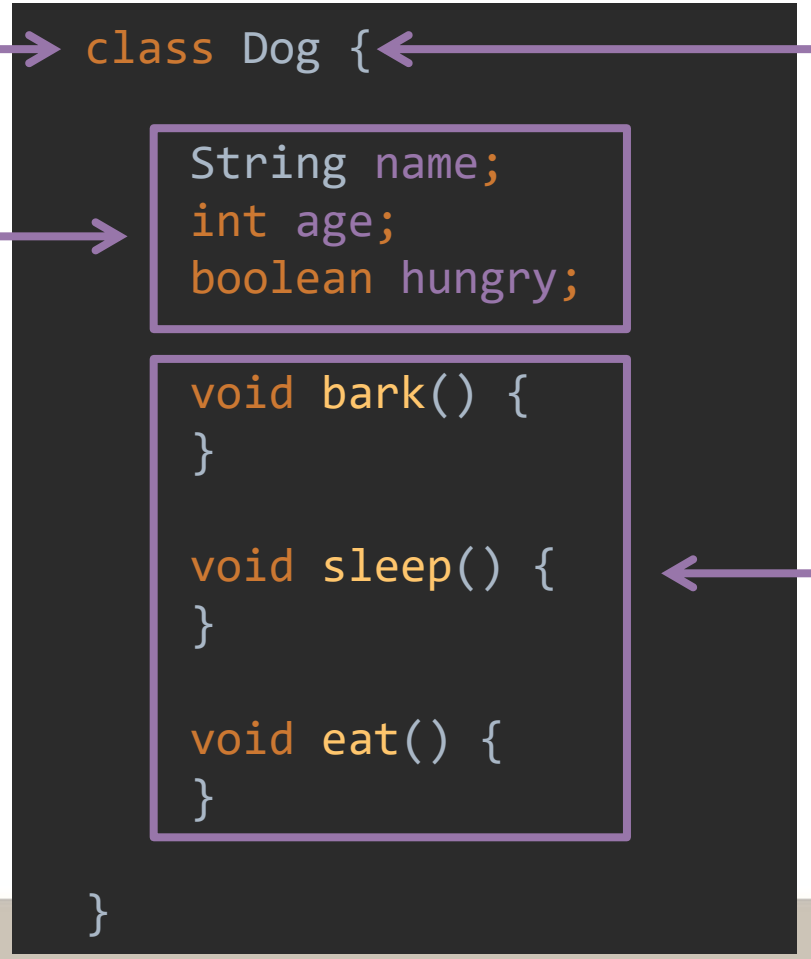
CLASS DECLARATION

```
class ClassName {

    type variable1;
    type variable2;
    ...
    type variableN;

    type method1() {}
    type method2() {}
    ...
    type methodN() {}

}
```

CLASS DECLARTION BREAKDOWN

Special keyword

Class variables
(fields) defining state



Class name

Class methods
defining behavior

OBJECT INSTANTIATION

- Object instantiation without assignment

```
new Class();
```

- Object instantiation with assignment

```
Class variable = new Class();
```

OBJECT INSTANTIATION

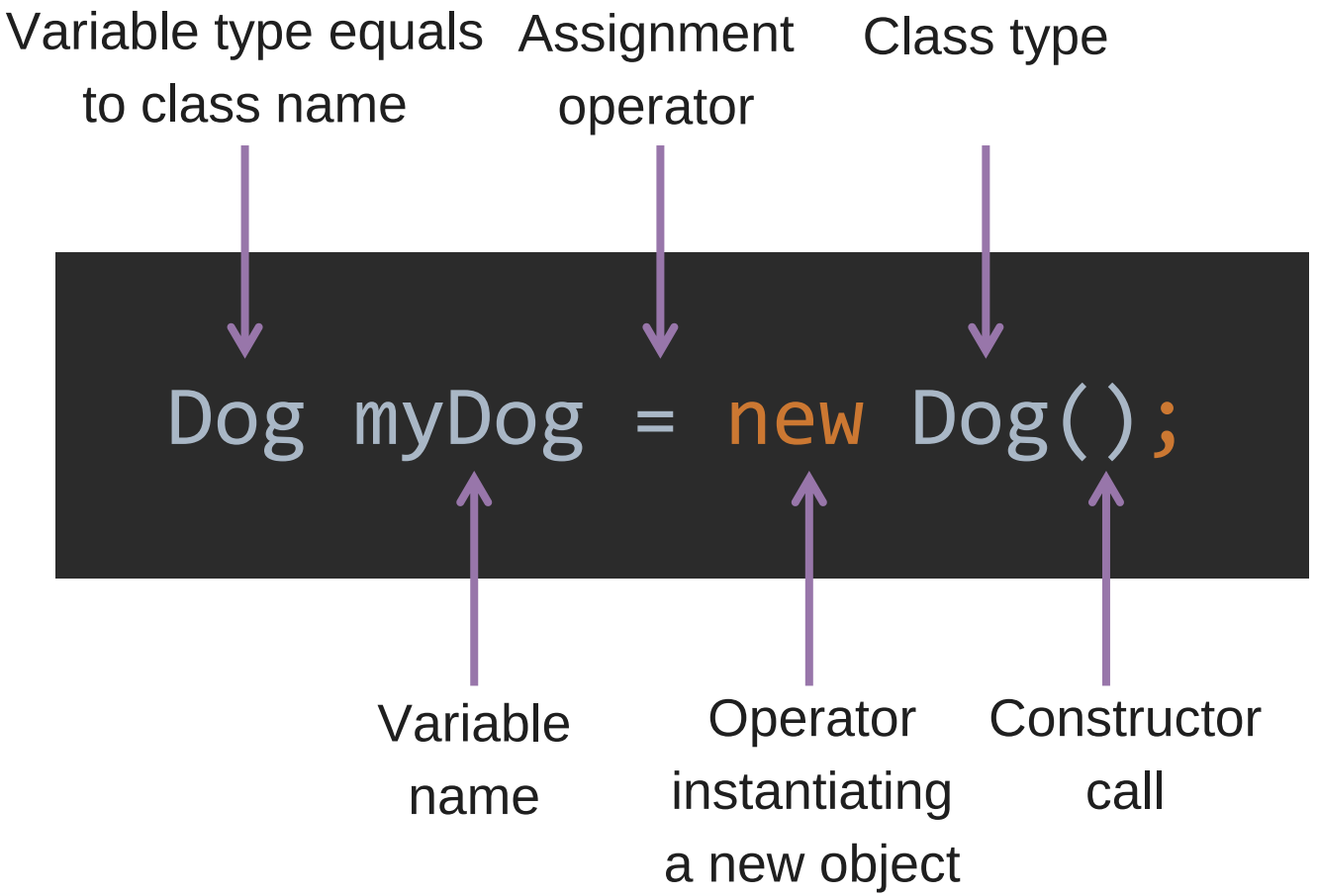
- Object instantiation without assignment

```
new Dog();
```

- Object instantiation with assignment

```
Dog myDog = new Dog();
```

OBJECT INSTANTIATION BREAKDOWN



CONCEPTS

1. Declaration - object variable declaration of a class type
2. Instantiation - the process of creating an object with new operator
3. Initialization - the process of object construction by setting its initial state

CONSTRUCTORS

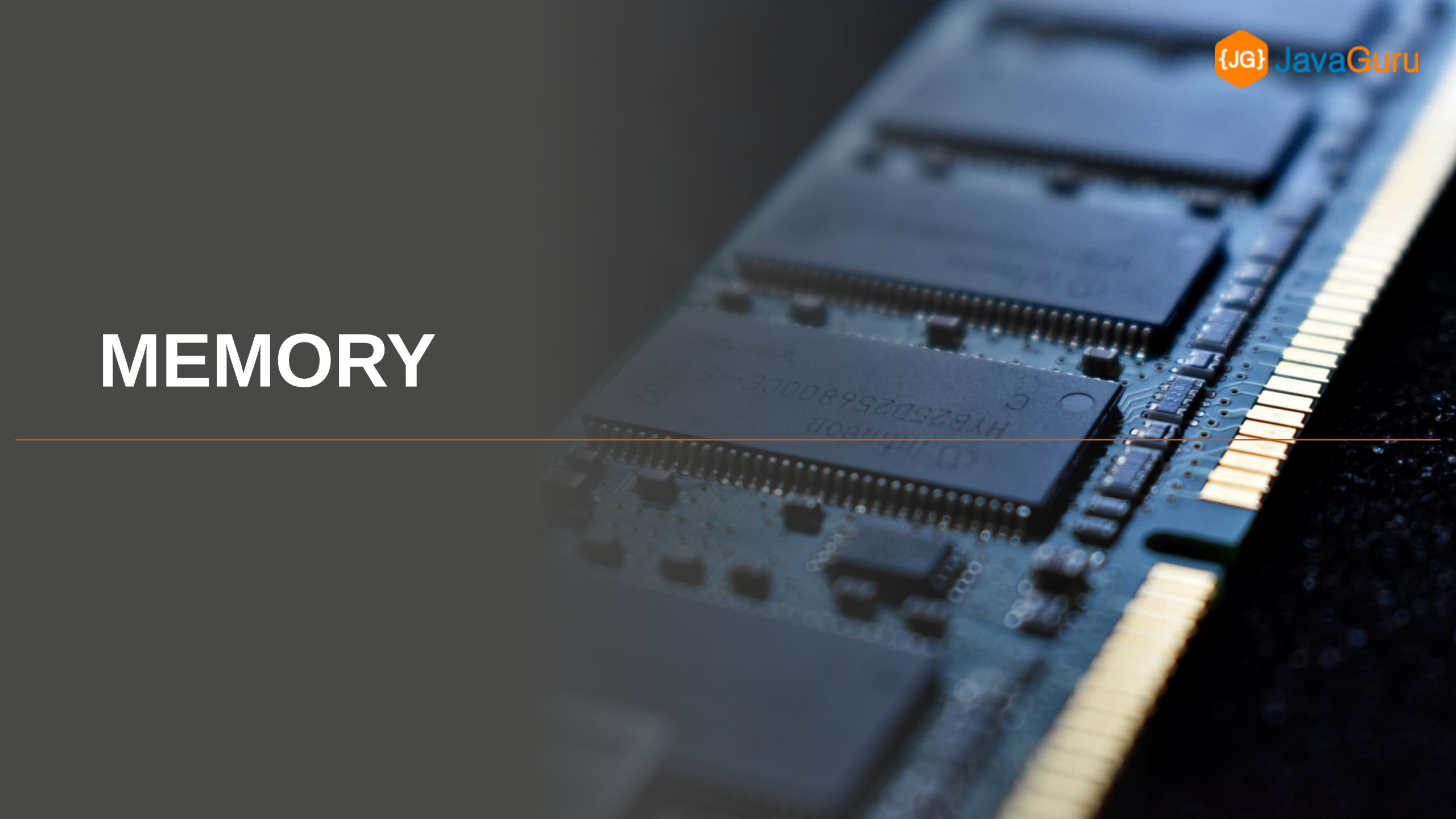
1. Every class has a constructor
2. If explicit constructor(s) is not specified in code, Java Compiler will generate default constructor implicitly
3. Each time a new object is created, at least one constructor will be invoked
4. Each defined constructor must have unique signature (i.e. ordered number and type of arguments)

CONSTRUCTOR DECLARATION

Explicit default constructor without arguments

```
class Dog {

    String name;

    Dog() {

    }

    Dog(String name) {
        this.name = name;
    }

}
```

Explicit constructor with argument and initialization



MEMORY

MEMORY TYPES

- Java Heap Memory
  - Created objects are stored in the heap space
  - Lives from the start till the end of application execution
  - Objects stored in heap are globally accessible
- Java Stack Memory
  - Contains local primitive variables and reference variables to objects in heap space
  - Lives only within method execution, short-lived
  - Bound to the current execution thread

METHODS

METHOD DEFINITION

- Java method is a collection of statements that are grouped together to perform an operation
  - Invoking *System.out.println()* method actually executes several statements in order to display a message on the console
- Describes behavior of class or actions that object can perform
- Method either produces output or not

METHOD DECLARATION

Defines the access type of the method

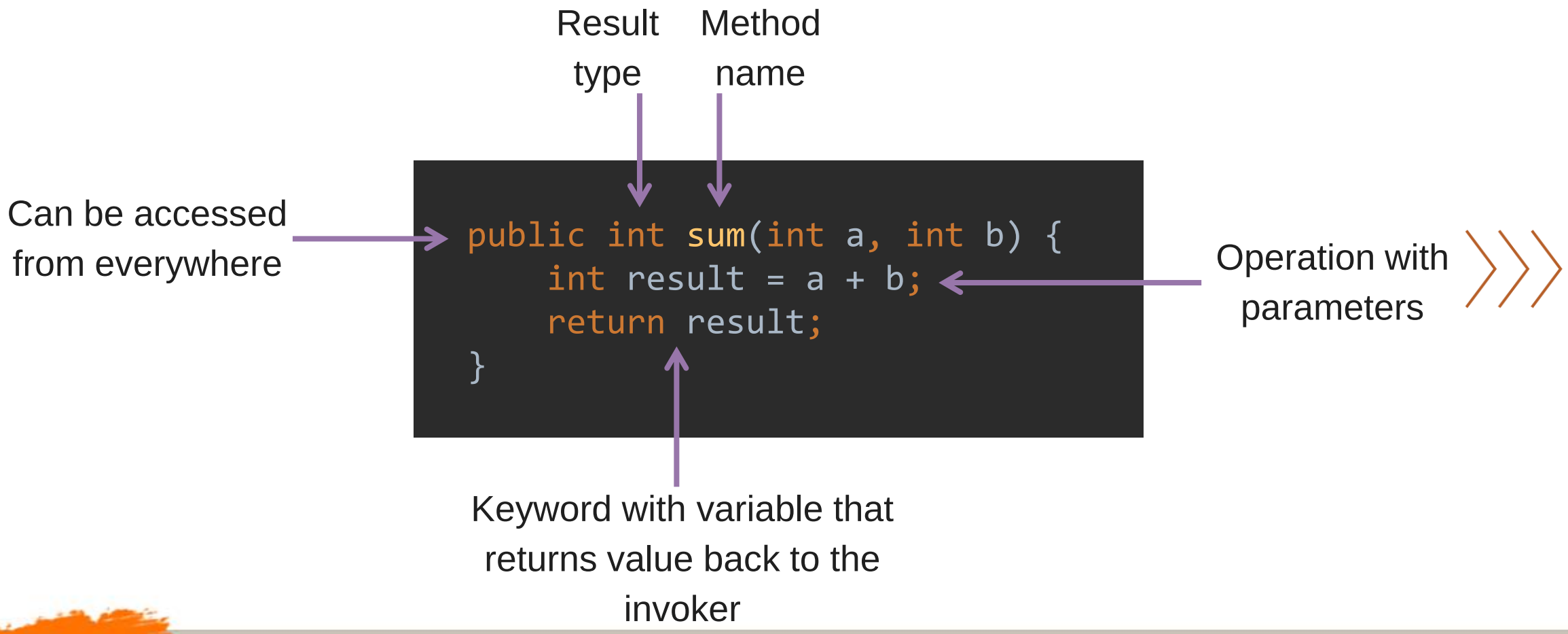
Defines method name

```
modifier returnType methodName(arg1, arg2,..., argN) {
    //body
}
```

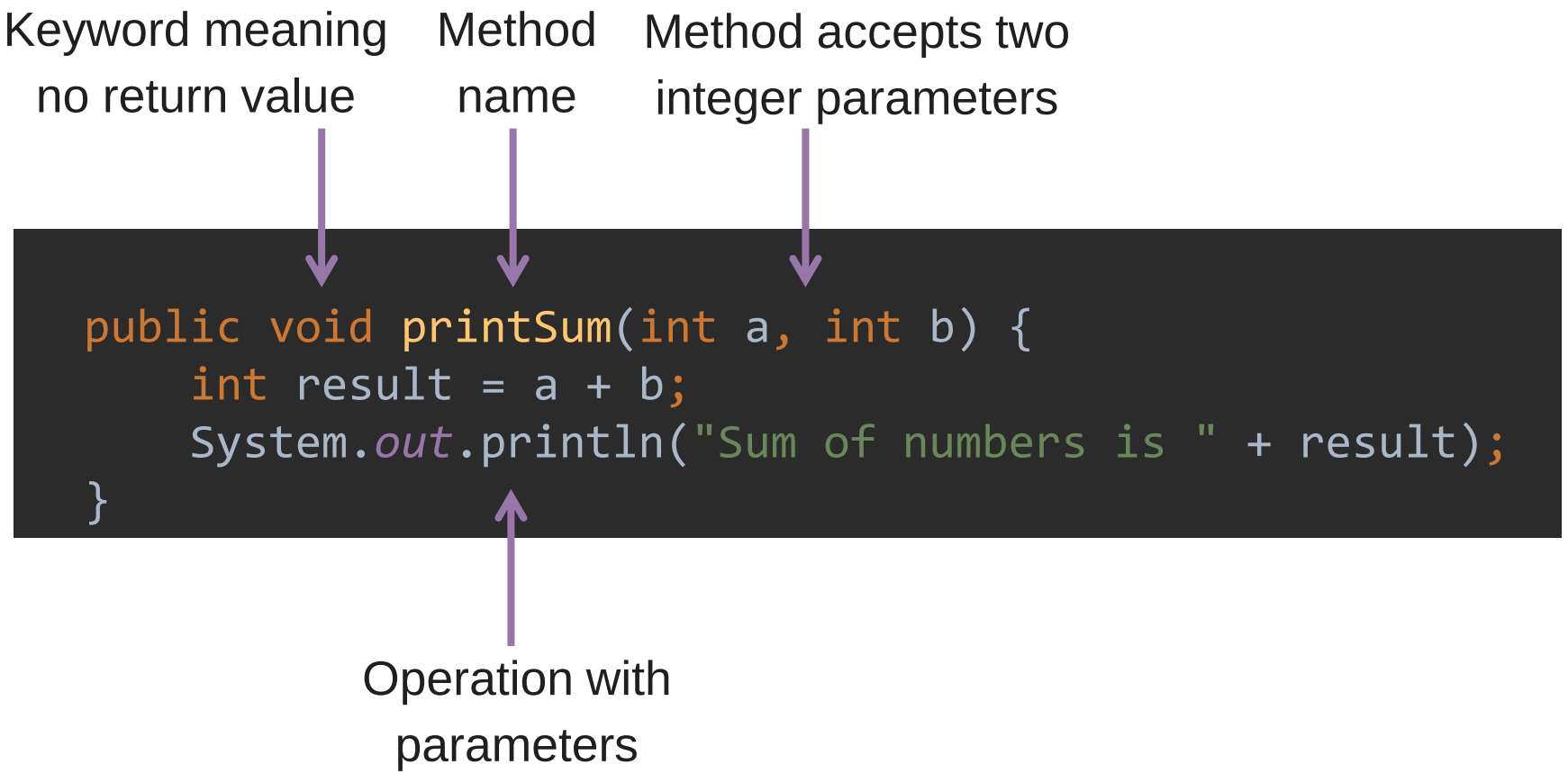
Specifies return type for methods producing output

List of parameters (type, name and order)

METHOD DECLARATION EXAMPLE



METHOD DECLARATION EXAMPLE



ABOUT RETURNING RESULT

- After completion method returns to the code that invoked it
- Whether method returns value or not is declared in method signature
- When type is void - return statement is unnecessary, however can be stated
- Other type - return statement is necessary

ACCESSING AND CHANGING OBJECT STATE

- In OOP another party should not be able to access object state directly
  - To keep things safe, one can
  - Retrieve object state via get methods (getters)
  - Change object state via set methods (setters)

GETTERS & SETTERS DECLARATION

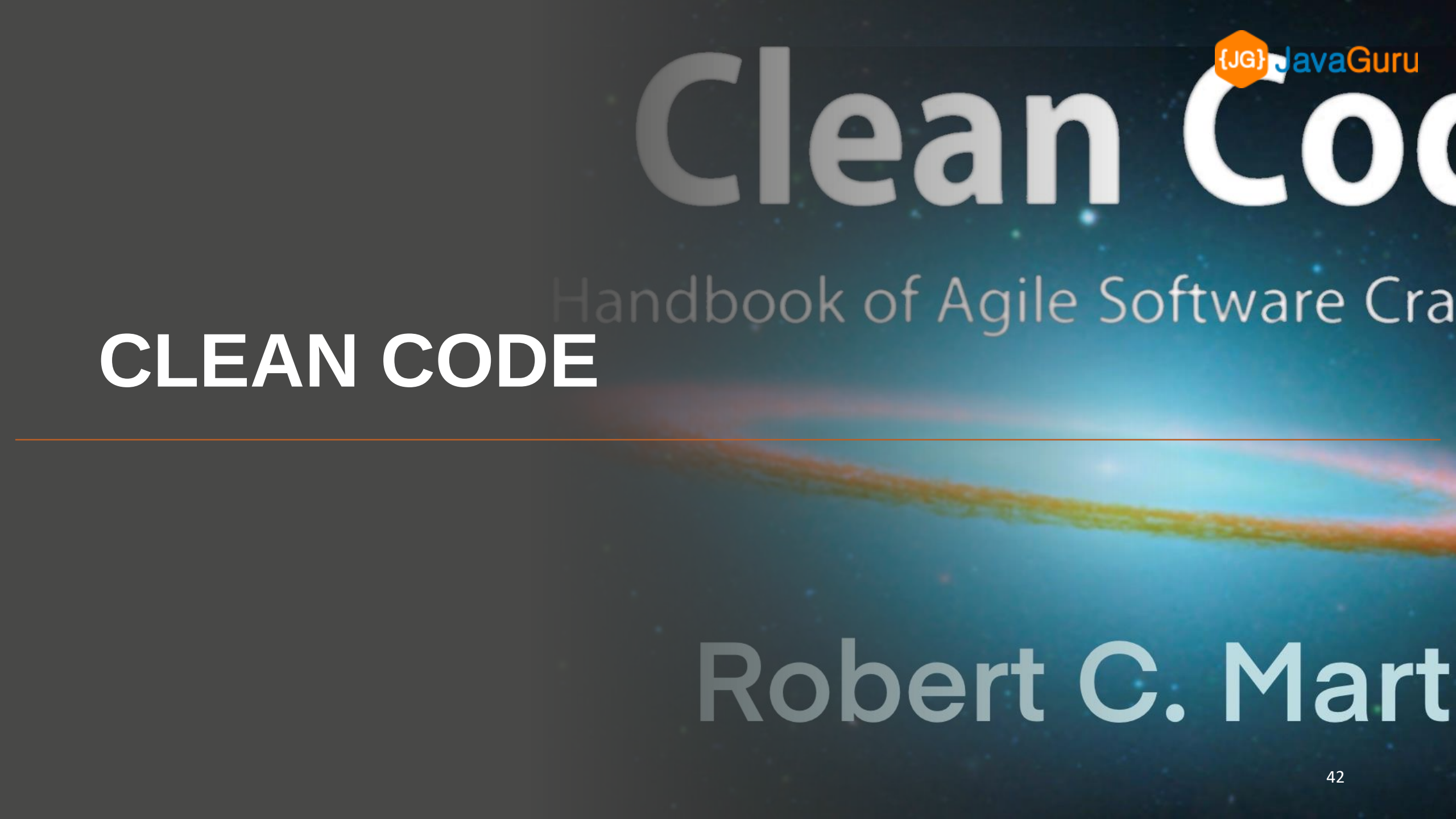
```
public class Person {

    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Getters

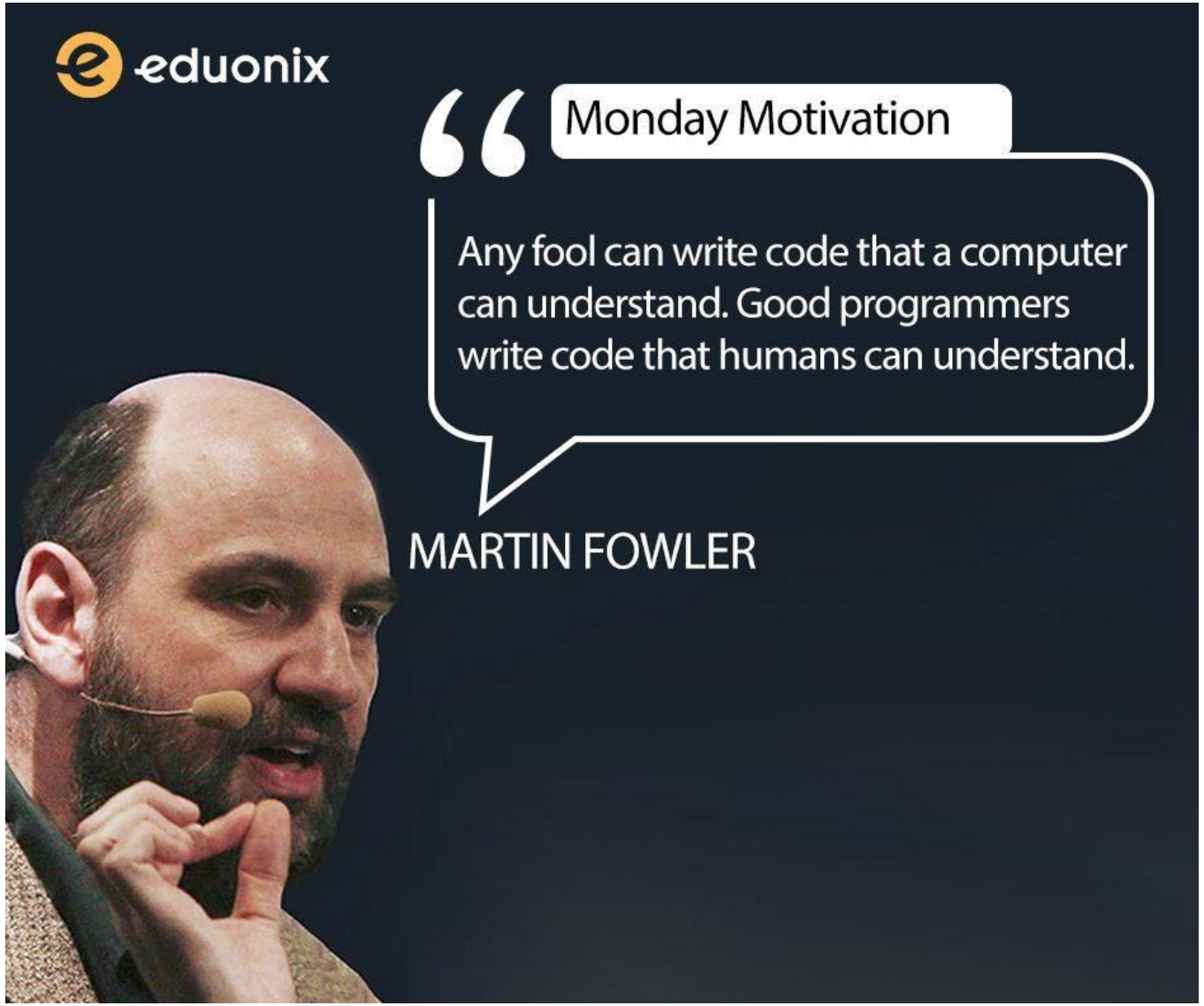
Setters

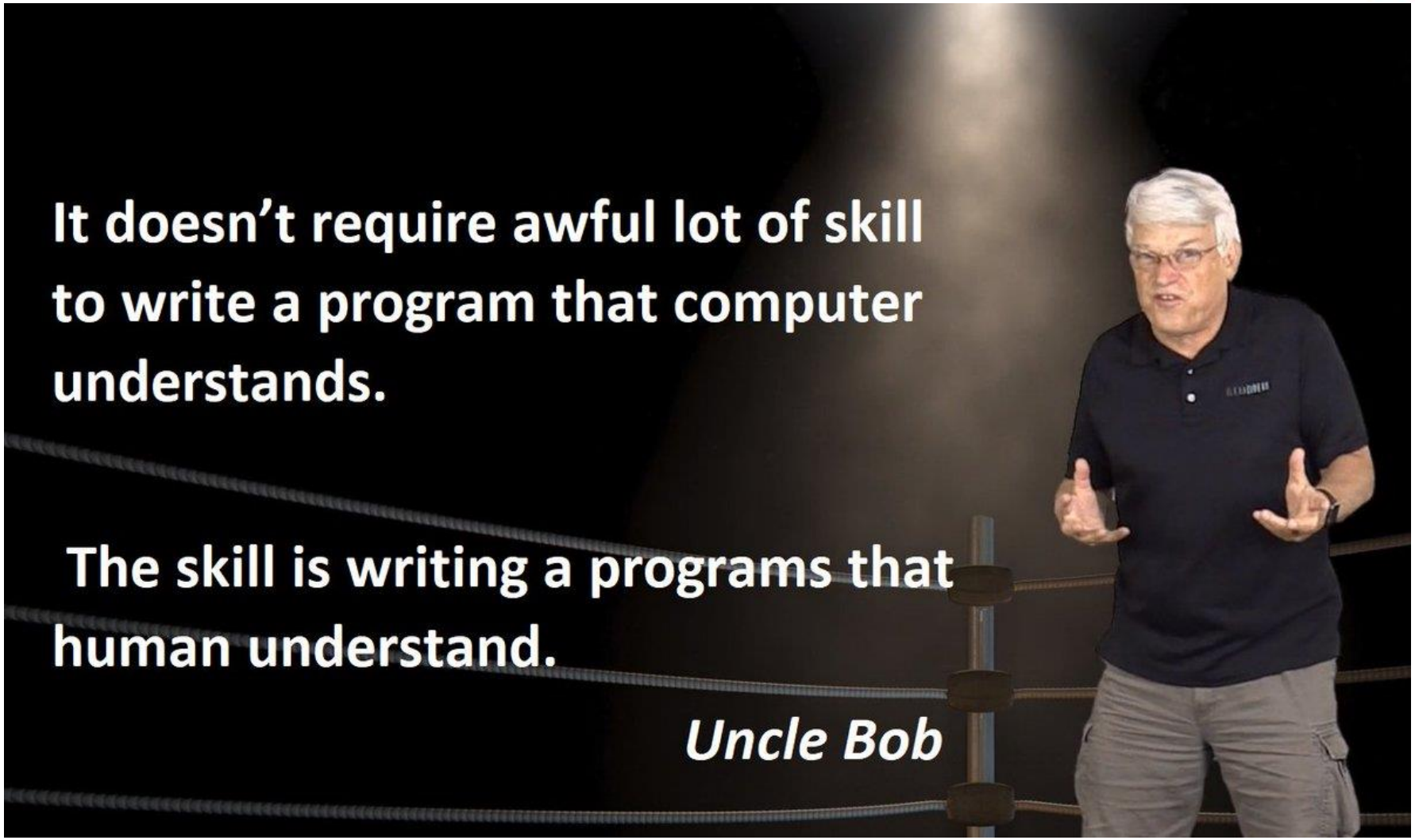
GETTERS & SETTERS USAGE

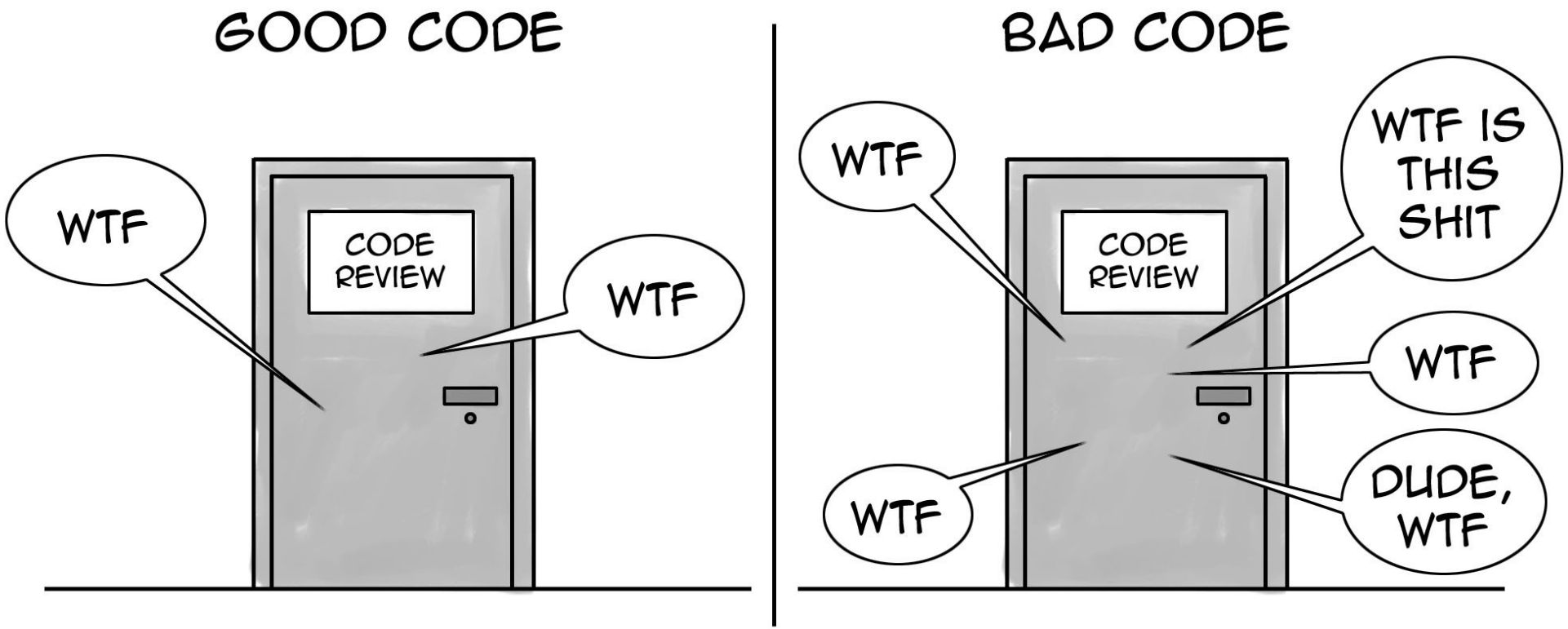
```
public class PersonTest {

    public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        person.setAge(32);

        String personName = person.getName();
        int personAge = person.getAge();

        System.out.println("His name is " + personName);
        System.out.println("He is " + personAge + " years old");
    }

}
```


{JG} JavaGuru
Clean Coo
Handbook of Agile Software Cra
Robert C. Mart

CLEAN CODE


eduonix
Monday Motivation
Any fool can write code that a computer can understand. Good programmers write code that humans can understand.
MARTIN FOWLER


It doesn't require awful lot of skill to write a program that computer understands.
The skill is writing a programs that human understand.
Uncle Bob



THE ONLY VALID MEASUREMENT OF CODE QUALITY: WTFS/MINUTE

BAD NAMING AND GOOD NAMING

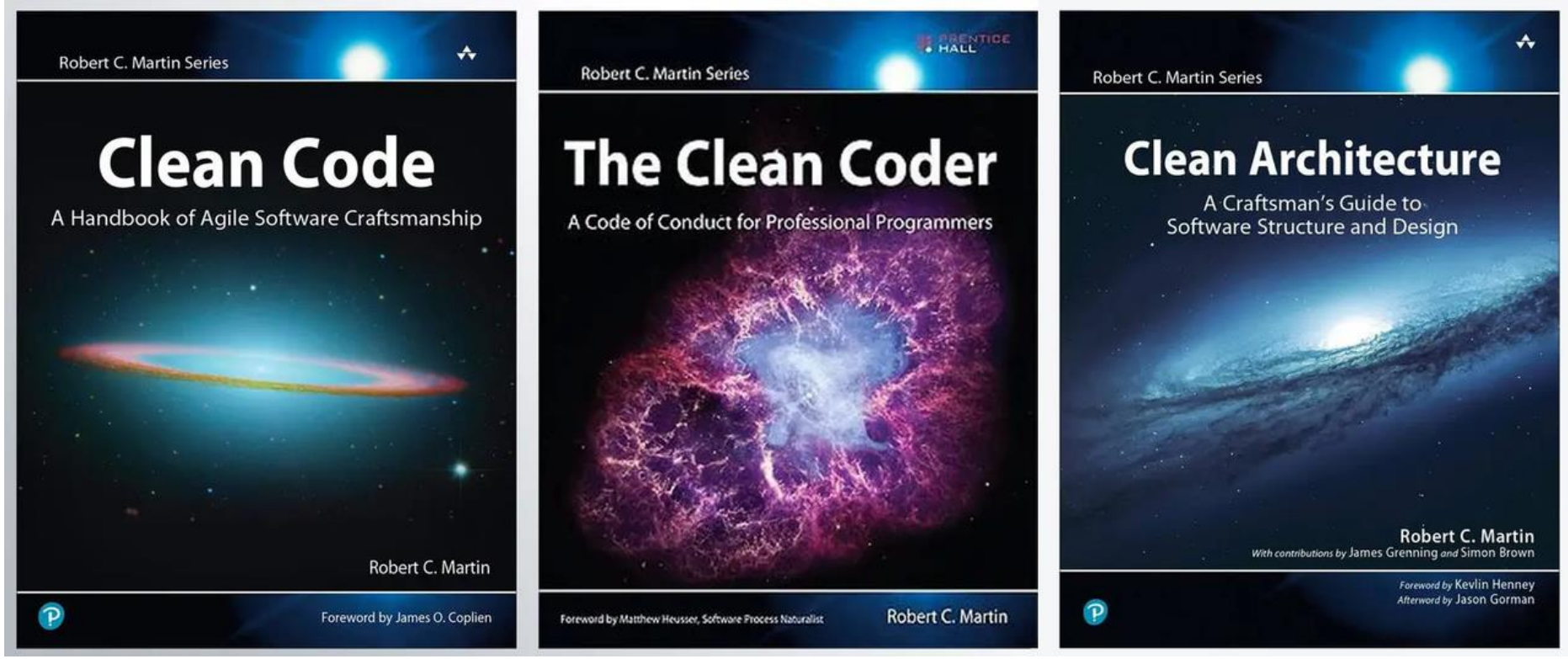
Bad Code

```
public class Cat {

    private String n;

    public String getN() {
        return n;
    }

    public void setN(String n) {
        this.n = n;
    }

    public void v() {
        System.out.println("Meow");
    }
}
```

Good Code

```
public class Cat {

    private String name;

    public String getName() {
        return n;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void voice() {
        System.out.println("Meow");
    }
}
```

CLEAN CODE BOOKS



REFERENCES


{JG} JavaGuru
POULTRY·BOOK
Northanger Abbey
JANE AUSTEN
D.H. LAWRENCE
Selected novels & stories
HAMLYN
Pilgrims Progress
1775
WALDEN

REFERENCES

- https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
- https://www.tutorialspoint.com/java/java_methods.htm


{JG} JavaGuru
QUESTIONS?

THANK YOU!