# System-Level Programming

## 3   Java/Python vs. C – Some Examples

**Peter Wägemann**

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

Summer Term 2025

http://sys.cs.fau.de/lehre/ss25

# The First C Program

- The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

03-Java-vs-C_en

# The First C Program

■ The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

■ Compilation and execution (on a UNIX system)

```
~> gcc -o hello hello.c
~> ./hello
Hello World!
~>
```

Not that complicated at all :-)

# The First C Program – a Comparison to Java

■ The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

■ The most famous program of the world in **Java**

```java
import java.lang.System;
class Hello {
  public static void main(String[] args) {
    /* greet user */
    System.out.println("Hello World!");
    return;
  }
}
```

# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

**C:** A C program starts with `main()`, a **global function** of type `int`, which is defined in exactly one **file**.

- The most famous program of the world in **Java**

```java
import java.lang.System;
class Hello {
  public static void main(String[] args) {
    /* greet user */
    System.out.println("Hello World!");
    return;
  }
}
```

**Java:** Each Java program starts with the function `main()`, a **static method** of type `void`, which is defined in exactly one **class**.

# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

**C:** <no counterpart>

- The most famous program of the world in **Java**

```java
import java.lang.System;
class Hello {
  public static void main(String[] args) {
    /* greet user */
    System.out.println("Hello World!");
    return;
  }
}
```

**Java:** Each Java program consists of at least one **class**.

# The First C Program – a Comparison to Java

■ The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

**C:** The output of the string takes place due to the **function** printf(). (\n ↝ new line)

■ The most famous program of the world in **Java**

```java
import java.lang.System;
class Hello {
  public static void main(String[] args) {
    /* greet user */
    System.out.println("Hello World!");
    return;
  }
}
```

**Java:** The output of one string takes place in the **method** println() from the class out, which is from the package System.

# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

**C:** To use the function `printf()`, the **library stdio.h** is included by the **preprocessor instruction** `#include`.

- The most famous program of the world in **Java**

```java
import java.lang.System;
class Hello {
  public static void main(String[] args) {
    /* greet user */
    System.out.println("Hello World!");
    return;
  }
}
```

**Java:** To use the **class out**, the **package System** is included by the `import` instruction.

# The First C Program – a Comparison to Java

- The most famous program of the world in **C**

```c
#include <stdio.h>

int main(int argc, char **argv) {
  /* greet user */
  printf("Hello World!\n");
  return 0;
}
```

C: Return to the operating system with **return value**. 0 in this case indicates that no error has happened.

- The most famous program of the world in **Java**

```java
import java.lang.System;
class Hello {
  public static void main(String[] args) {
    /* greet user */
    System.out.println("Hello World!");
    return;
  }
}
```

Java: Return to the operating system.

03-Java-vs-C_en

# A Comparison to Hello World in Python

- The most famous program of the world in **Python**

```python
print('Hello World')
```

# A Comparison to Hello World in Python

- The most famous program of the world in **Python**

```
print('Hello World')
```

- **Python**
    - Usually execution in interpreter
    - High-level abstraction level
    - Numerous built-in functions
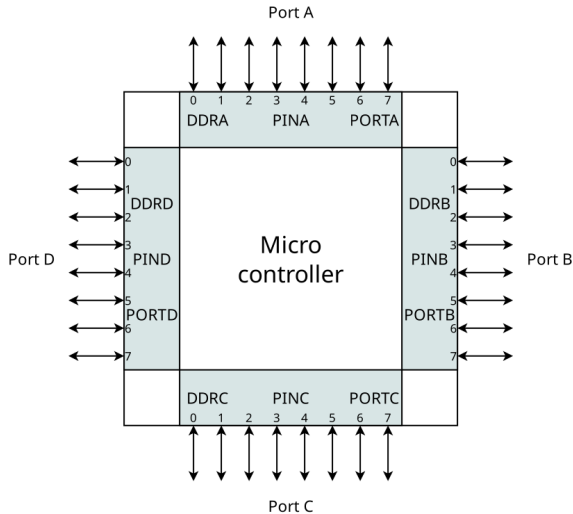    - No explicit include directive required for built-in function
- **C**
    - Native execution on machine
    - No interpreter required
    - Machine orientation
    - Explicit include statements for used functions

# The First C Program for a µController
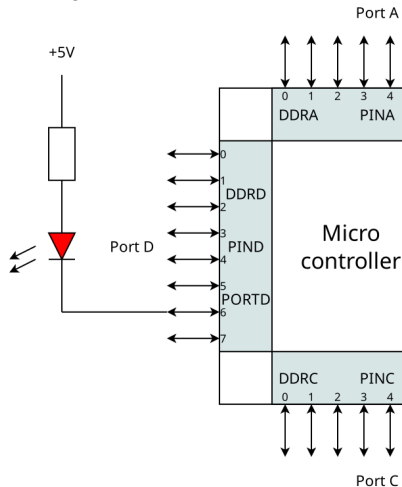
Preliminary information:



- DDRx: data direction register
- PINx: port input register
- PORTx: port output register (of size 8 bit each)

# The First C Program for a µController

Background information:



- LED is not lit:
  - DDRD bit 6: '1' (output)
  - PORTD bit 6: '1' (5V)
- LED lights up:
  - DDRD bit 6: '1' (output)
  - PORTD bit 6: '0' (0V)

03-Java-vs-C__en

# The First C Program for a µController

- "Hello world" for AVR ATmega (SPiCboard)

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

03-Java-vs-C_en

# The First C Program for a µController

- "Hello world" for AVR ATmega (SPiCboard)

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

- Compilation and **flashing** (with SPiC-IDE)           ⤳ Exercises

03-Java-vs-C_en

# The First C Program for a µController

- "Hello world" for AVR ATmega (SPiCboard)

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

- Compilation and **flashing** (with SPiC-IDE)                    ↝ Exercises

- Execution (SPiCboard):   🔴⚪🟢🟣🔴🟡🟢🔵                    (red LED lit)

03-Java-vs-C_en

# The First C Program for a µController

- "Hello world" for AVR ATmega (SPiCboard)

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

µController programming is "somewhat different".

- Compilation and **flashing** (with SPiC-IDE)  ↝ Exercises

- Execution (SPiCboard):    (red LED lit)

03-Java-vs-C_en

# The First C Program for a µController

- "Hello world" for AVR ATmega (compare ↪ 3–3)

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

The `main()`-function has **no return value** (type `void`). A µController program runs **indefinitely** ⤳ `main()` does not terminate.

03-Java-vs-C_en

ⓒ klsw    System-Level Programming (ST 25)    3 Java/Python vs. C – Output    3–7

# The First C Program for a µController

- "Hello world" for AVR ATmega (compare ↪ 3–3 )

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

There will be **no return** to an operating system
(which one?). The endless loop assures that `main()`
does not terminate.

# The First C Program for a µController

- "Hello world" for AVR ATmega (compare ↪ 3–3)

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

First, the **hardware** is initialized (i.e., put in a predefined state). For this, **single bits** in certain **hardware registers** have to be changed.

# The First C Program for a µController

- "Hello world" for AVR ATmega (compare ↪ 3–3 )

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

> The interaction with the environment (in this case: switching on the LED) takes place by **manipulating single bits** in hardware registers.

# The First C Program for a μController

- "Hello world" for AVR ATmega (compare ↪ 3–3 )

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1<<6);  /* PD6 is used as output */
  PORTD |= (1<<6);  /* PD6: high --> LED is off */

  /* greet user */
  PORTD &= ~(1<<6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

To access the hardware registers (DDRD, PORTD, provided as **global variables**), the **library** avr/io.h is included with #include.

03-Java-vs-C_en

# The Second C Program – Input with Linux

■ user interaction (reading one character) with Linux:

```c
#include <stdio.h>

int main(int argc, char **argv) {
  printf("Press key: ");
  char key = getchar();

  printf("You pressed %c\n", key);
  return 0;
}
```

# The Second C Program – Input with Linux

■ user interaction (reading one character) with Linux:

```c
#include <stdio.h>

int main(int argc, char **argv) {
  printf("Press key: ");
  char key = getchar();

  printf("You pressed %c\n", key);
  return 0;
}
```
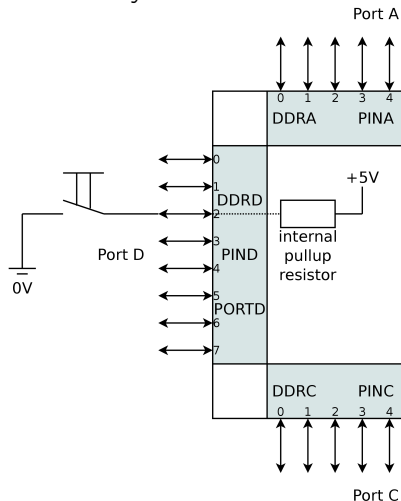
The getchar()-function reads one charac-
ter from the standard input (here: keyboard).
The function "waits", if necessary, until a cha-
racter is available.

# The Second C Program for a µController

Preliminary information:



- Initialising:
  - DDRD bit 2: '0' (input)
  - PORTD bit 2: '1' (pull-up switched on)
- Detection:
  - PIND bit 2: '1' => button not pressed
  - PIND bit 2: '0' => button pressed

03-Java-vs-C_en

# The Second C Program – Input with µController

- User interaction (waiting for a button to be pressed) on the SPiCboard:

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: button on port D pin 2 */
  DDRD  &= ~(1 << 2);  /* PD2 is used as input */
  PORTD |=  (1 << 2);  /* activate pull-up: PD2: high */

  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1 << 6);  /* PD6 is used as output */
  PORTD |= (1 << 6);  /* PD6: high --> LED is off */

  /* wait until PD2 -> low (button is pressed) */
  while ((PIND >> 2) & 1) {
  }

  /* greet user */
  PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

- User interaction (waiting for a button to be pressed) on the SPiCboard:

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: button on port D pin 2 */
  DDRD  &= ~(1 << 2);  /* PD2 is used as input */
  PORTD |=  (1 << 2);  /* activate pull-up: PD2: high */

  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1 << 6);  /* PD6 is used as output */
  PORTD |= (1 << 6);  /* PD6: high --> LED is off */

  /* wait until PD2 -> low (button is pressed) */
  while ((PIND >> 2) & 1) {
  }

  /* greet user */
  PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

Just like the LED, the button is connected to a **digital IO pin** of the µController. We now configure pin 2 at port D as an **input** by **clearing** the corresponding bits in the register DDRD.

03-Java-vs-C_en

# The Second C Program – Input with µController

User interaction (waiting for a button to be pressed) on the SPiCboard:

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: button on port D pin 2 */
  DDRD  &= ~(1 << 2);  /* PD2 is used as input */
  PORTD |=  (1 << 2);  /* activate pull-up: PD2: high */

  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1 << 6);  /* PD6 is used as output */
  PORTD |= (1 << 6);  /* PD6: high --> LED is off */

  /* wait until PD2 -> low (button is pressed) */
  while ((PIND >> 2) & 1) {
  }

  /* greet user */
  PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

By **setting** bit 2 in the register `PORTD` as 1, the internal pull-up resistor (high resistance) is activated. Which is connected to $V_{CC} \rightsquigarrow$ PD2 = *high*.

03-Java-vs-C_en

- User interaction (waiting for a button to be pressed) on the SPiCboard:

```c
#include <avr/io.h>

void main(void) {
  /* initialize hardware: button on port D pin 2 */
  DDRD  &= ~(1 << 2);  /* PD2 is used as input */
  PORTD |=  (1 << 2);  /* activate pull-up: PD2: high */

  /* initialize hardware: LED on port D pin 6, active low */
  DDRD  |= (1 << 6);  /* PD6 is used as output */
  PORTD |= (1 << 6);  /* PD6: high --> LED is off */

  /* wait until PD2 -> low (button is pressed) */
  while ((PIND >> 2) & 1) {
  }

  /* greet user */
  PORTD &= ~(1 << 6); /* PD6: low --> LED is on */

  /* wait forever */
  while (1) {
  }
}
```

> **Active waiting:** waits for a button to be pressed, i.e., while PD2 (bit 2 in the register PIND) is *high*. When the button is pressed, PD2 is pulled to ground ⇝ bit 2 in the register PIND is now *low* and the loop is exited.

```java
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
  private JFrame frame;

  public static void main(String[] args) {
    // create input, frame and button objects
    Input input = new Input();
    input.frame = new JFrame("Java Program");
    JButton button = new JButton("Press me");

    // add button to frame
    input.frame.add(button);
    input.frame.setSize(400, 400);
    input.frame.setVisible(true);

    // register input as listener of button events
    button.addActionListener(input);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("Button pressed!");
    System.exit(0);
  }
}
```

Input as a "typical" Java program (**object-oriented**, **graphic**)

```java
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
  private JFrame frame;

  public static void main(String[] args) {
    // create input, frame and button objects
    Input input = new Input();
    input.frame = new JFrame("Java Program");
    JButton button = new JButton("Press me");

    // add button to frame
    input.frame.add(button);
    input.frame.setSize(400, 400);
    input.frame.setVisible(true);

    // register input as listener of button events
    button.addActionListener(input);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("Button pressed!");
    System.exit(0);
  }
}
```

Input as a "typical" Java program (**object-oriented**, **graphic**)

The class **Input** implements an **interface** to receive interaction events.

03-Java-vs-C_en

ⓒ klsw    System-Level Programming (ST 25)    3 Java/Python vs. C  – Input    3–12

```java
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
  private JFrame frame;

  public static void main(String[] args) {
    // create input, frame and button objects
    Input input = new Input();
    input.frame = new JFrame("Java Program");
    JButton button = new JButton("Press me");

    // add button to frame
    input.frame.add(button);
    input.frame.setSize(400, 400);
    input.frame.setVisible(true);

    // register input as listener of button events
    button.addActionListener(input);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("Button pressed!");
    System.exit(0);
  }
}
```

Input as a "typical" Java program (**object-oriented**, **graphic**)

The program behaviour is implemented with the help of a multitude of **objects** (frame, button, input), which are created during initialization.

## As a Reference: User Interaction as a Java-Program

```java
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
  private JFrame frame;

  public static void main(String[] args) {
    // create input, frame and button objects
    Input input = new Input();
    input.frame = new JFrame("Java Program");
    JButton button = new JButton("Press me");

    // add button to frame
    input.frame.add(button);
    input.frame.setSize(400, 400);
    input.frame.setVisible(true);

    // register input as listener of button events
    button.addActionListener(input);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("Button pressed!");
    System.exit(0);
  }
}
```

Input as a "typical" Java program (**object-oriented**, **graphic**)

The created `button`-object sends a message to the `input`-object.

03-Java-vs-C_en

ⓒ klsw    System-Level Programming (ST 25)    3 Java/Python vs. C – Input    3–12

# As a Reference: User Interaction as a Java-Program

```java
import java.lang.System;
import javax.swing.*;
import java.awt.event.*;

public class Input implements ActionListener {
  private JFrame frame;

  public static void main(String[] args) {
    // create input, frame and button objects
    Input input = new Input();
    input.frame = new JFrame("Java Program");
    JButton button = new JButton("Press me");

    // add button to frame
    input.frame.add(button);
    input.frame.setSize(400, 400);
    input.frame.setVisible(true);

    // register input as listener of button events
    button.addActionListener(input);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("Button pressed!");
    System.exit(0);
  }
}
```

Input as a "typical" Java program (**object-oriented**, **graphic**)

The button press is signaled by an `actionPerformed()`-message (method call).

03-Java-vs-C_en

■ The program cannot be compared to its counterpart in C directly.
  - It uses the (already known to you) **object-oriented paradigm**, which is typical for Java.
  - This difference shall be emphasised here.

■ User interaction in Java explained line by line
  The button press is signaled by an `actionPerformed()`-message (method call).

# 1$^{st}$ Takeaway: Java/Python → C (Syntax)

- Java/Python and C have similar **syntax**
  (Syntax: "What do valid programs of the language look like?")

- C syntax was used as a reference for the development of
  Java/Python
  ↝ many language elements are similar or identical
  - blocks, loops, conditions, statements, literals
  - these elements will be looked at in detail in the following chapters

- Major elements from Java/Python are **not** present in C
  - classes, packages, objects, exceptions, . . .

# 1st Takeaway: Java/Python → C (Idiomatic)

- There are major **idiomatic** differences
(Idiomatic: "What do programs of the language usually look like?")

- **Java: object-oriented paradigm**
  - Central question: From which things is a problem made of?
  - Segmentation of the problem in **classes** and objects
  - Hierarchy by inheritance and aggregation
  - Program flow by interaction between objects
  - Re-usability through extensive class libraries

- **C: imperative paradigm**
  - Central question: From which steps is the problem made up?
  - Segmentation of the problem in **functions** and variables
  - Hierarchy by breakdown into functions
  - Program flow through calls between functions
  - Re-usability through function libraries

# 1$^{st}$ Takeaway: Java/Python → C (Philosophy)

- There are **philosophical** differences as well
  (Philosophy: "Basic ideas and concepts of a language")
- **Java:** Security and portability due to **abstracting from machine**
  - Compilation for virtual machines (JVM)
  - Extensive checks for programming errors during runtime
    - range overflow, division by 0, . . .
  - Problem-centric memory model
    - Only type-safe memory accesses, automatic garbage collection during runtime.
- **C:** efficiency and lightweight due to **machine orientation**
  - Compilation for concrete hardware architecture
  - No checks for programming errors during runtime
    - some error are caught by the operating system − if present
  - Memory model directly maps to the machine
    - pointers provide direct memory access
    - coarse-grained access protection and automatic garbage collection
      (at processor level) by an OS − if present

# A First Takeaway: µController Programming

> ### C ↦ machine orientation ↦ µC programming
> The machine orientation of the language C especially shows when looking at µController programming!

- Only one program is running
  - On RESET the program is loaded directly from flash memory
  - Hardware has to be initialized by the program first
  - Shall never terminate (e.g., with the help of a infinite loop in `main()`)

- The solution is implemented close to the machine
  - Direct manipulation of single bits in hardware registers
  - Therefore detailed knowledge of *electrical wiring* is needed
  - No support of an operating system (like Linux)
  - Usually a low level of abstraction ⤳ error-prone... *but fast*

# A First Takeaway: µController Programming

> ### C ↦ machine orientation ↦ µC programming
> The machine orientation of the language C especially shows when looking at µController programming!

- Only one program is running
  - On RESET the program is loaded directly from flash memory
  - Hardware has to be initialized by the program first
  - Shall never terminate (e. g., with the help of a infinite loop in `main()`)

- The solution is implemented close to the machine
  - Direct manipulation of single bits in hardware registers
  - Therefore detailed knowledge of *electrical wiring* is needed
  - No support of an operating system (like Linux)
  - Usually a low level of abstraction ⤳ error-prone... *but fast*

  **Approach:** Higher abstraction with problem-oriented libraries

03-Java-vs-C_en