Software Engineering WiSe 25/26

Assignment 1: Program Abstractions

Hand-In via Moodle: Oct. 29, 10:00

- This assignment provides up to **one** bonus point.
- This assignment must be submitted as a **group** via Moodle. One team member submits it on behalf of the team.
- Submit **one PDF** with the solution for all tasks (font size 12).
- The PDF must start with a **title page** that mentions all group members their names, student ids, and email addresses. Invalid submissions are desk-rejected, i.e., zero points without further checking or resubmissions.
- Please use appropriate tools to create the solution (e.g., LibreOffice/Word for texts or draw.io for graphics). Scanned handwritten solutions will only be accepted if they are very legible.
- If you have any screenshots/pictures, make sure they are in high resolution.
- Provide well-formulated, but concise and specific answers to the tasks (no bulletpoints).
- Any kind of plagiarism is prohibited, as well as the use of generative AI (e.g., ChatGPT) to create the final content of this hand-in (except for non-essential content, such as illustrative images as long as it properly credited).

1. Team Contract

Define what a team member is expected to do to be considered a good team member. Your definition will be a contract between the team members, so be sure to make it explicit so that all members understand and agree to the statements you include in your contract. Discuss the goals and objectives of the team. These could be related to both a grade and learning outcomes. The important thing is that the team has a common understanding of what to achieve and that you have a negotiated level of ambition. There is no need to hand in the contract via Moodle.

2. Abstract Syntax Tree

Your task is to create an abstract syntax tree (AST) for the Java code below. Begin by identifying the parts of the code that end up in an AST, such as class declarations, method declarations, variable declarations, and control structures. Create a hierarchical representation of the code's structure using nodes to represent different parts, and edges to denote nesting relationships and order of elements. Label the nodes with the AST types given in the lecture. If a Java element was not listed there, create a descriptive name yourself (or look it up from the Java language specification). An IF-ESLE statement is represented as a node with a condition and the different branches as child nodes.

Be sure to capture the nesting and order of operations accurately in your AST. Creating a detailed and accurate AST will demonstrate your understanding of the code's structure and how it can be represented as a tree data structure.

Hints: you may use available tools or draw the AST by hand.

```
1  //...
2  {
3      int x = 17;
4      while(x >= 0) {
5          if(x % 20 == 0) {
6              x = x - 20;
7          } else{
8              x = x - 5;
9          }
10         if(x % 10 == 0) {
11             x = x - 10;
12         }
13     }
14     print(x);
15 }
16 //...
```

3. Control-Flow Graph (Code to CFG)

Consider the Java code snippet below. Your task is to construct a control-flow graph (CFG) that represents the code's control flow. The CFG should consist of nodes that represent the basic blocks, which group various program instructions (e.g., statements, declarations), and edges that represent the control flow between these blocks. Identify the entry and exit points of the CFG and include them in your diagram. Annotate each node in the CFG with the corresponding code line number (or code range) instead of the full code. Make sure to capture all possible flows. Constructing a detailed CFG will help you understand the flow of control in the code and provide insights into its structure and behavior.

```
1  public void doStuff() {
2      int x = 10;
3      int y = 20;
4      if (x > y) {
5          System.out.println("x is greater than y");
6          x--;
7      } else if (x < y) {
8          System.out.println("x is less than y");
9          while (x <= y) {
10             x = x * 2;
11         }
12     } else {
13         System.out.println("x is equal to y");
14     }
15     System.out.println("x + y = " + (x + y));
16 }
```

Hint: You may use software tools or draw the CFG and intermediate representation by hand. Providing the intermediate representation is not necessary to receive the full score.

4. Control-Flow Graph (CFG to Code)

Consider the Control-Flow Graph (CFG) below. Your task is to provide a valid Java function those control flow matches the given CFG. Also, provide the matching between source code and the CFG, e.g., by mapping the node ids to the source code lines. Providing one possible implementation is sufficient for this task.