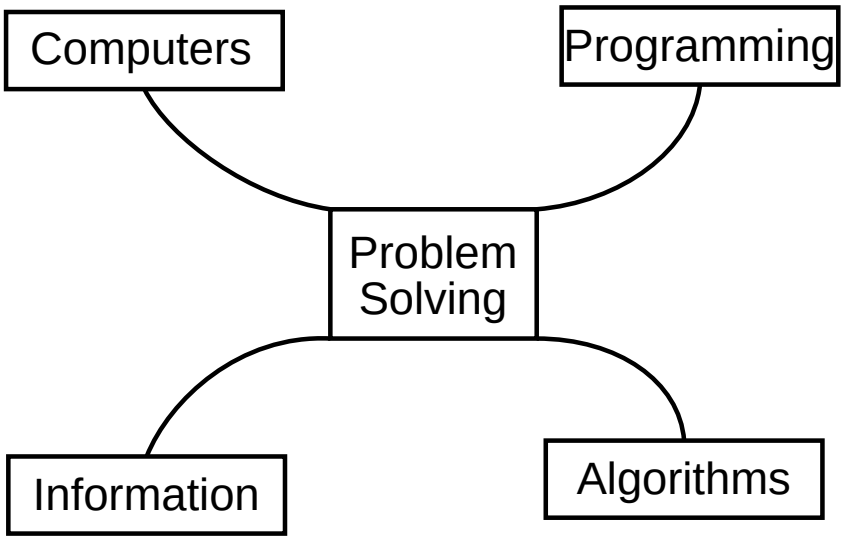
Announcements

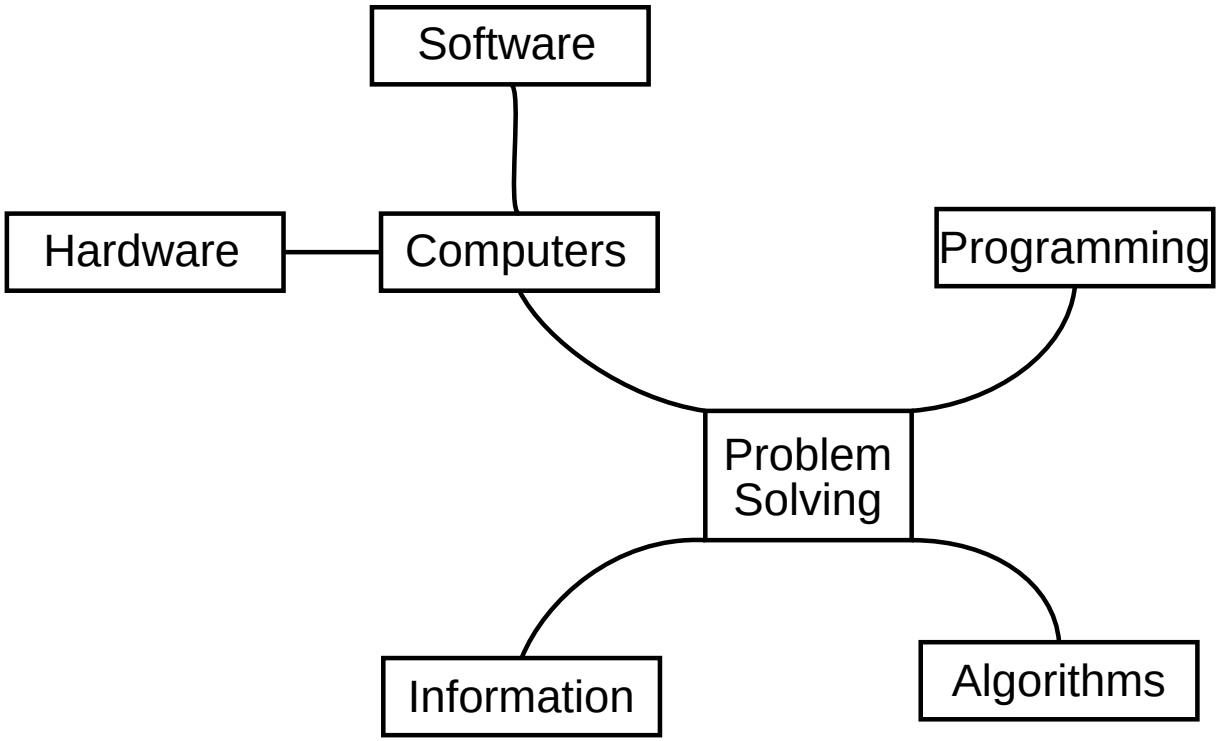
- Windows and Unix machines at the Trottier lab
- Account creation only for Unix

Road map

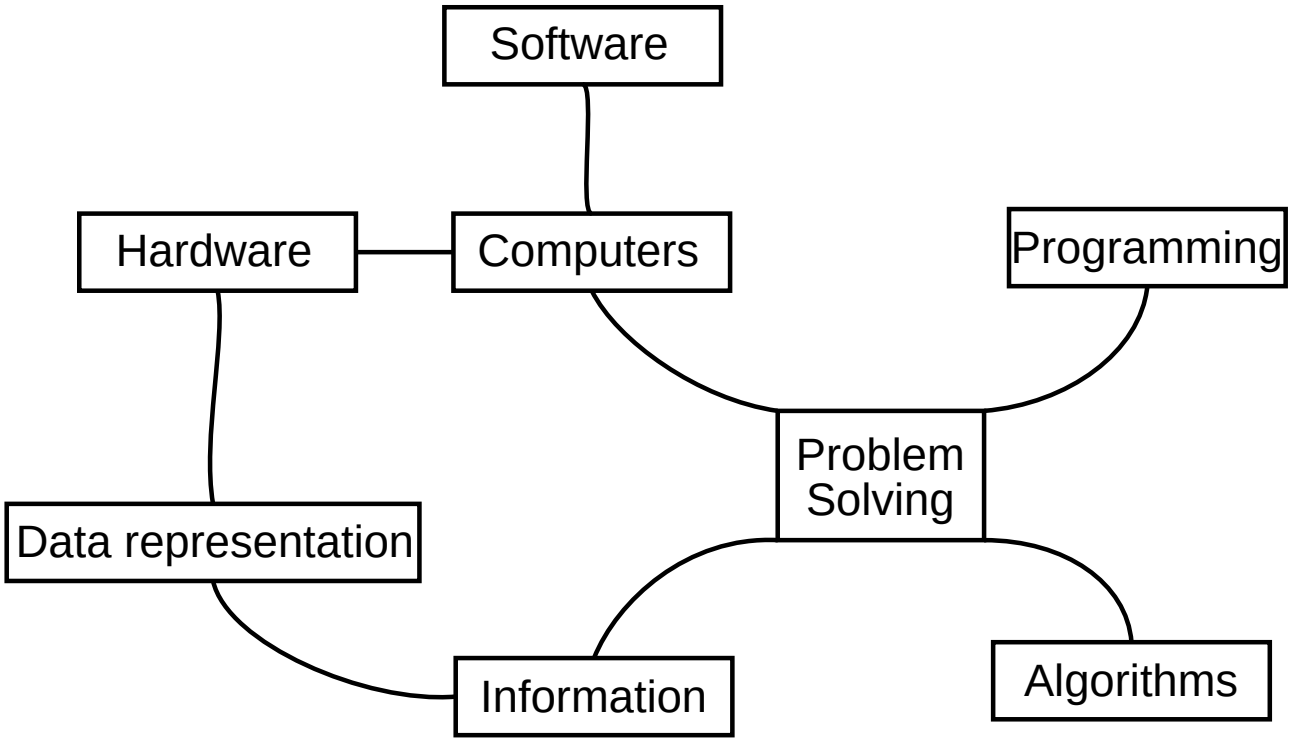
Road map


Computers
Programming
Problem Solving
Information
Algorithms

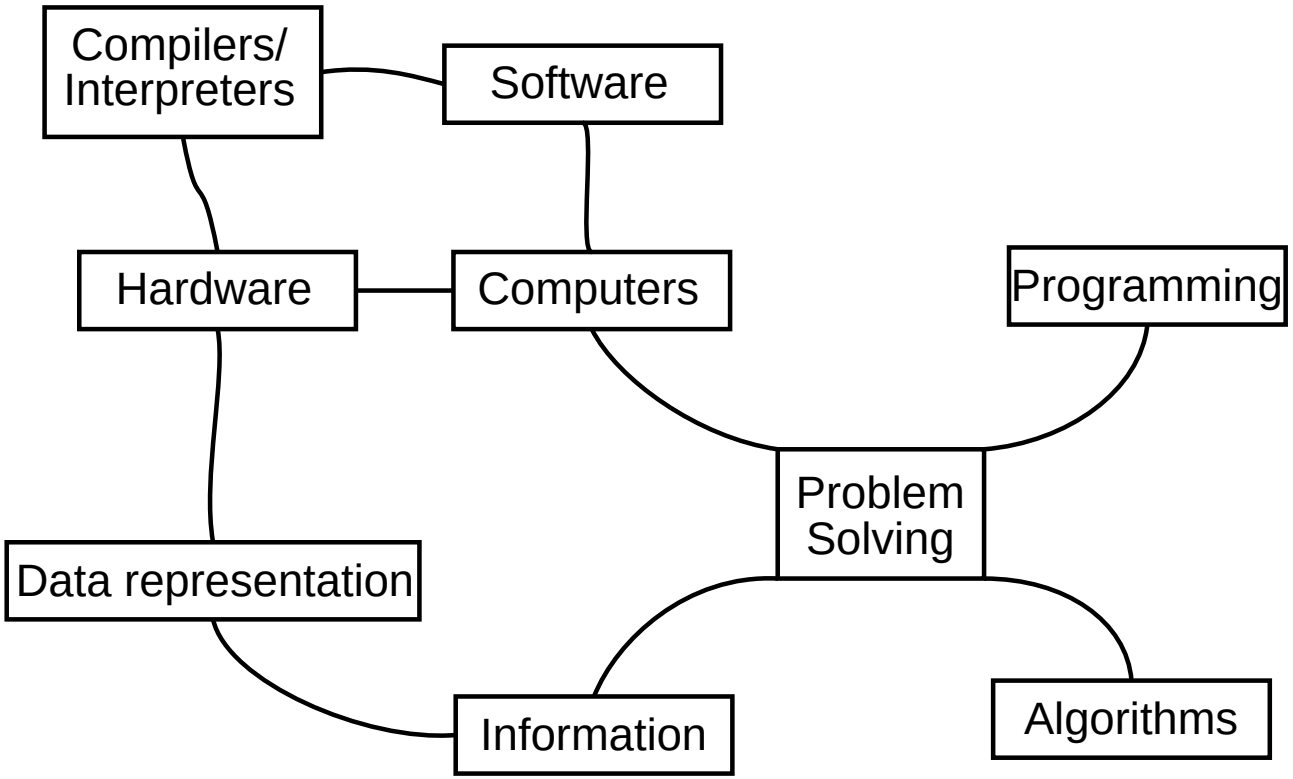
Road map


Software
Hardware
Computers
Programming
Problem Solving
Information
Algorithms

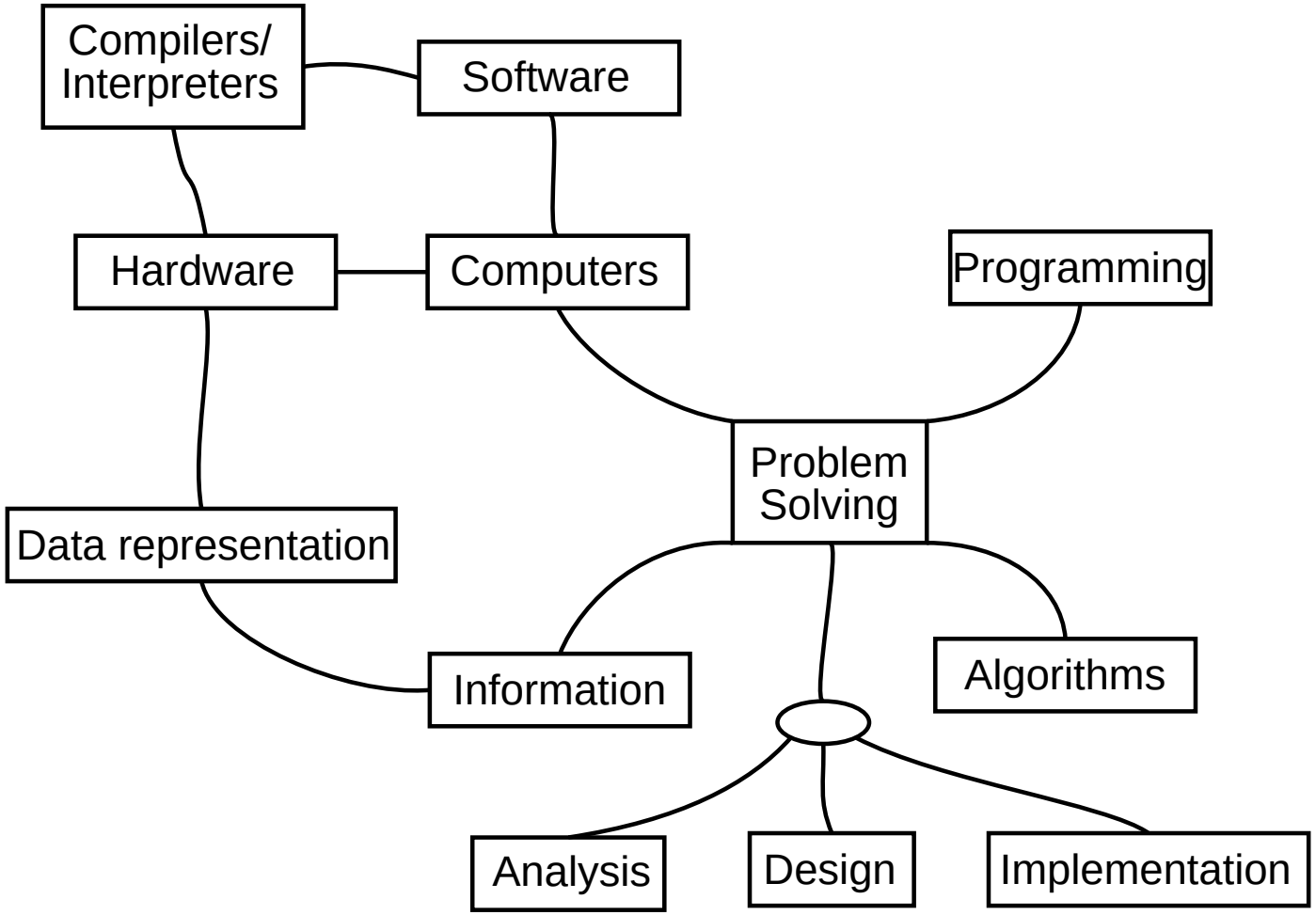
Road map


Software
Hardware
Computers
Programming
Problem Solving
Data representation
Information
Algorithms

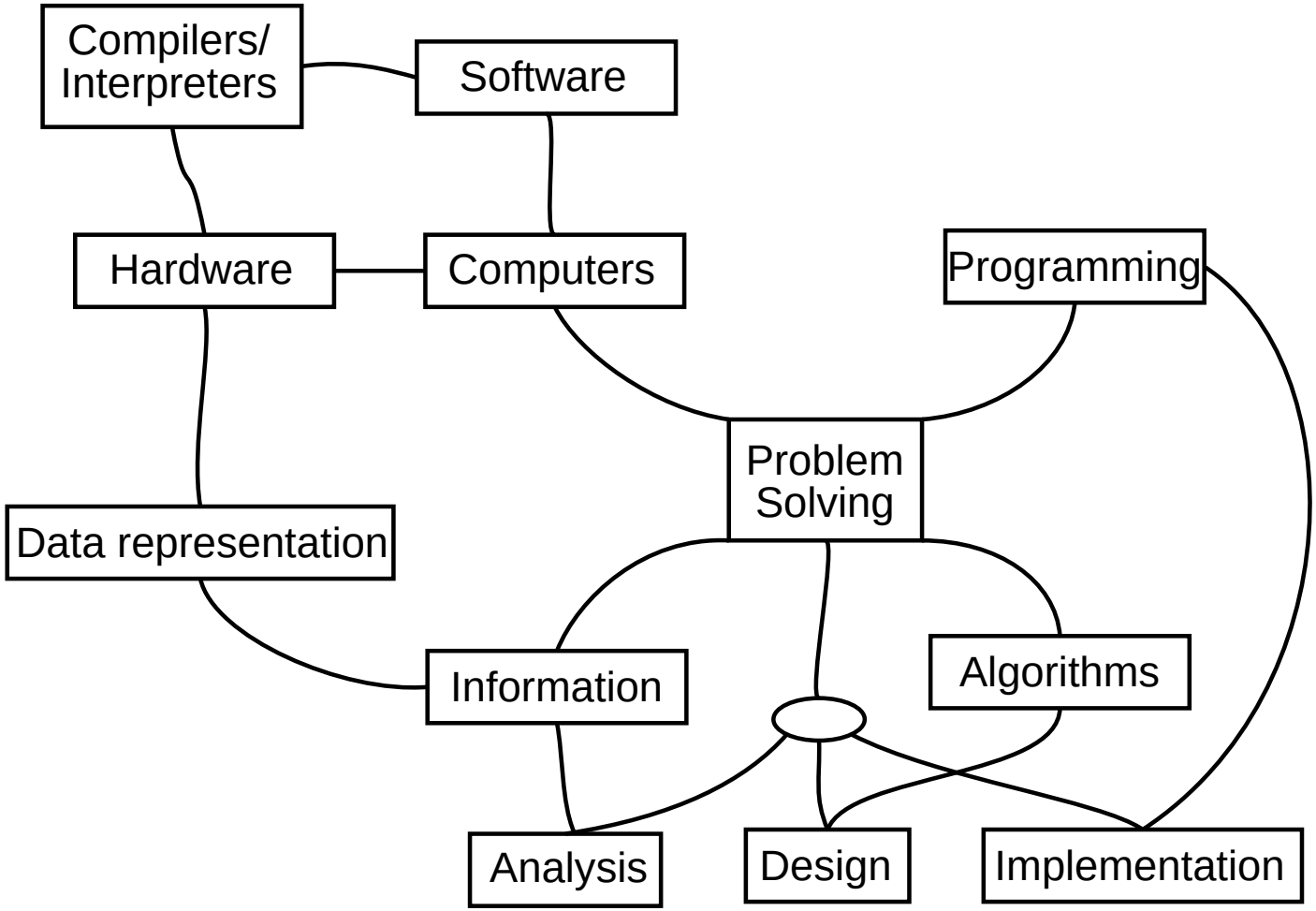
Road map


Compilers/
Interpreters
Software
Hardware
Computers
Programming
Problem
Solving
Data representation
Information
Algorithms

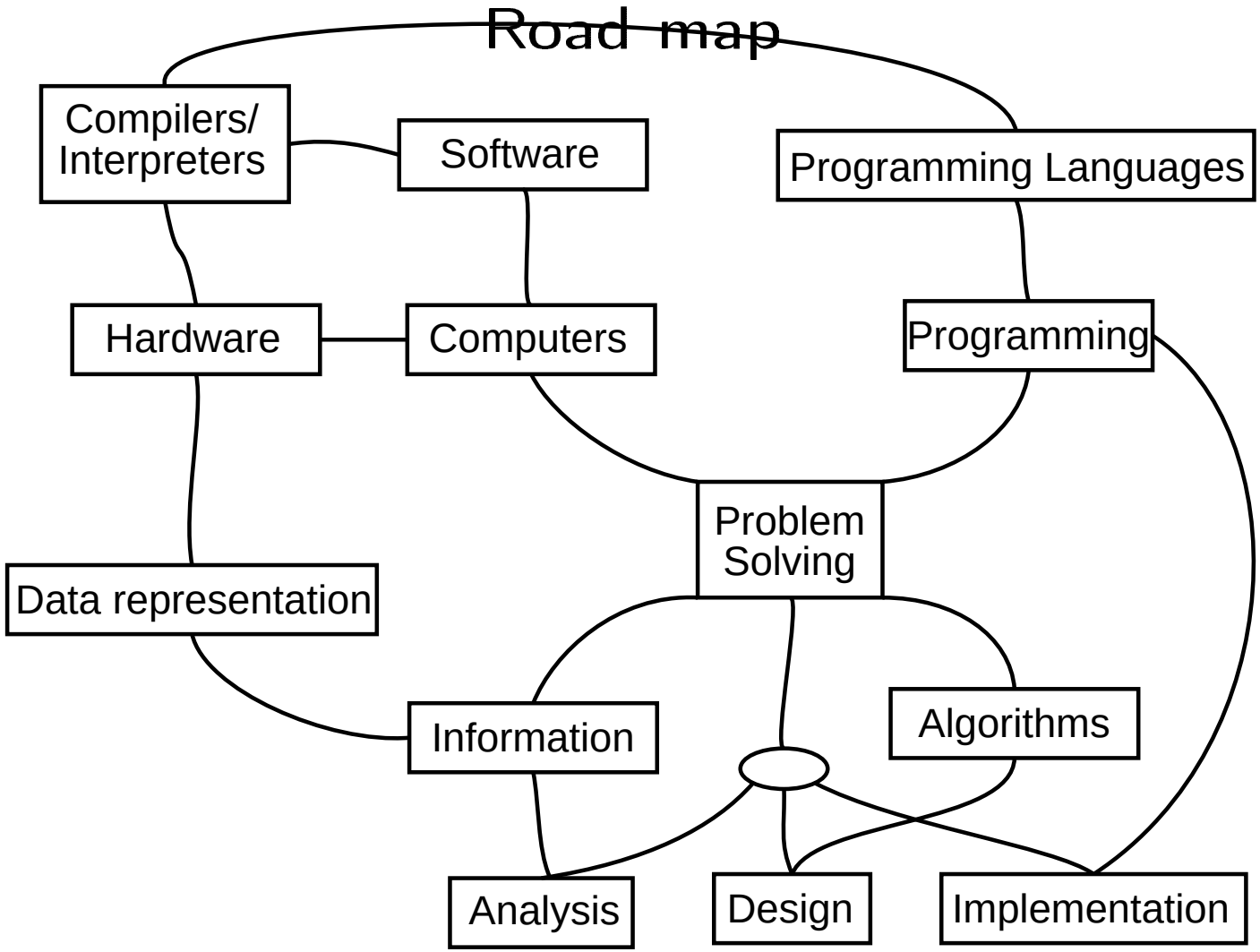
Road map


Compilers/
Interpreters
Software
Hardware
Computers
Programming
Problem
Solving
Data representation
Information
Algorithms
Analysis
Design
Implementation

Road map


Compilers/
Interpreters
Software
Hardware
Computers
Programming
Problem
Solving
Data representation
Information
Algorithms
Analysis
Design
Implementation

Road map


Compilers/
Interpreters
Software
Programming Languages
Hardware
Computers
Programming
Problem
Solving
Data representation
Information
Algorithms
Analysis
Design
Implementation

Programming Languages

- Machine language (binary, processor dependent)
- Assembly language (textual, low-level, processor dependent)
- High-level languages (textual, abstract, processor independent)
  - There are many high-level languages: Java, C, C++, C#, ML, Haskell, Scheme, Prolog, Python, Perl, etc.
  - Different types of languages:
    - Imperative
      - Procedural
      - Object Oriented
      - Concurrent
    - Declarative:
      - Functional
      - Logic
    - Mixed

Executing programs

- Editing
- Compilation/Interpretation
  - (Native) Compilation: Translation to machine language + Execution
    - Advantages: Fast, processor specific code is generated
    - Disadvantage: Needs a compiler for each type of processor; generates a different target file for each type of processor
  - Interpretation: Direct execution
    - Advantages: Execution is processor independent. Does not generate a different target file for each possible processor (portability)
    - Disadvantage: Slow execution due to overhead of interpretation.
  - Combined: Translation to bytecode + interpretation of bytecode
    - Best of both worlds: Only one file is generated (portable) and it is faster to execute than direct interpretation (but slower than native compilation.)

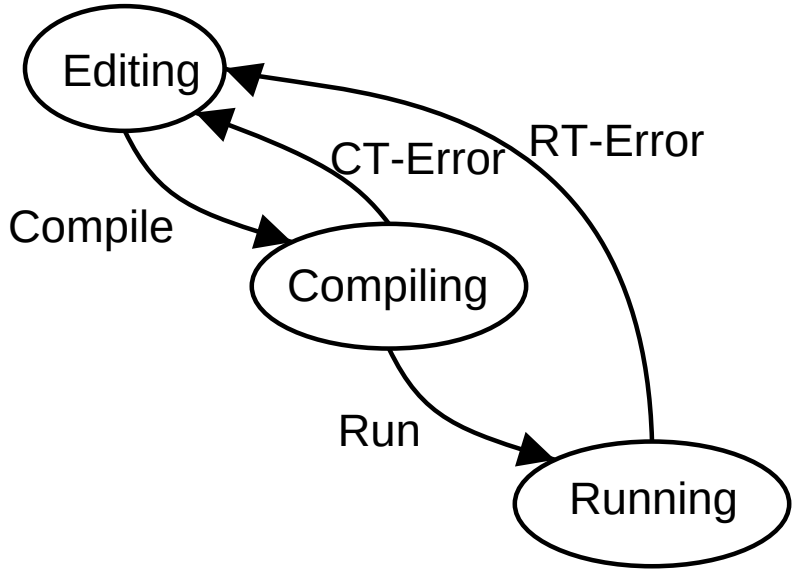
Languages

- Elements of a language
  - Alphabet
  - Syntax (grammar)
  - Semantics (meaning)
- Elements of Java:
  - Alphabet of Java: ASCII
  - Syntax: 'constructs'
    - Class definitions
    - Method definitions
    - Statements
    - others
  - Semantics: computation

Errors

- Errors:
  - Compile-time errors
  - Run-time errors
    - Exceptions
    - Logical

Errors



Basic Java Syntax

- A Java program is made up of one or more *class definitions*
- A class definition is made up of zero or more *method definitions*
- A method definition is made up of zero or more *statements* and *variable declarations*
- Roles:
  - Classes: Modules and Types of objects
  - Methods: procedures, functions, algorithms
  - Statements: instructions

Basic Java Syntax

```
public class ClassName
{
    // Body of ClassName
    // ...
    // List of method definitions
}
```

Basic Java Syntax

```
public class HelloWorld
{
   // Body of ClassName
   // ...
   // List of method definitions
}
```

Basic Java Syntax

```
public class Classname
{
    // method header
    {
        // method body: list of statements
    }
}
```

Basic Java Syntax

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello");
        System.out.println("Good bye");
    }
}
```

Basic Java Syntax

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println(“Hello”);
        System.out.println(“Good bye”);
    }
}
```

Bad Java Syntax

```
public class HelloWorld
{
    System.out.println(“Hello”);
    System.out.println(“Good bye”);
}
```

Bad Java Syntax

```
public static void main(String[] args)
{
    System.out.println(“Hello”);
    System.out.println(“Good bye”);
}
```

Bad Java Syntax

```
public static void main(String[] args)
{
    public class HelloWorld
    {
        System.out.println(“Hello”);
        System.out.println(“Good bye”);
    }
}
```

Indentation

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println(“Hello”);
        System.out.println(“Good bye”);
    }
}
```

Indentation

```
public class HelloWorld
{
public static void main(String[] args)
{
System.out.println(“Hello”);
System.out.println(“Good bye”);
}
}
```

Indentation

```
                public class HelloWorld
{
 public static void main(String[] args)
                                    {
        System.out.println(“Hello”);
System.out.println(“Good bye”);
    }
                   }
```

Indentation

```
public class HelloWorld{public static void main(St
ring[] args){System.out.println(“Hello”);System.ou
t.println(“Good bye”);}}
```

User Interface

- The user interface of a program is the way it interacts with the user: keyboard/mouse/windows/text

- Graphical User Interface:
  - Windows: buttons, text boxes, slidebars, graphics, etc.
  - Input with mouse and keyboard.

- Textual User Interface:
  - Console window: plain text
  - Input: keyboard only
  - Output:

```
System.out.println("text");
```

Introduction to statements

- The print statement

```
System.out.println(string_literal);
System.out.print(string_literal);
```

- String literals:

```
"(almost)any characters"
"This is a string literal"
"String literals can contain almost any character,
"a"
""
"24"
```

Introduction to statements

- String concatenation:

  string_literal + *string_literal*
  string_literal + *number_literal*

  “This is a ”+“message”
  “This is a message”
  “There are ”+70+“ students in this class”

- String literals with numbers are not numbers: “17” is not the same as 17

  “17” + “29”

is

  “1729”

while

  17 + 29

is

  46

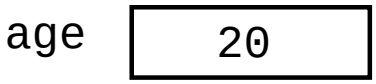
Simple programs

```
// File: PrintingStuff.java
public class PrintingStuff
{
    public static void main(String[] args)
    {
        System.out.println("This trivial program j
        System.out.println("prints this text to a
        System.out.println("Window.");
    }
}
```

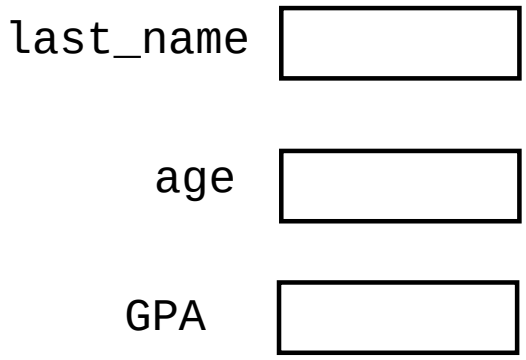
Variables

- A variable is a memory location
- A variable can contain information
- A variable has a symbolic name

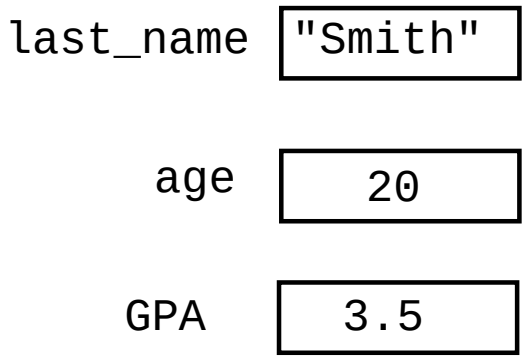
Variables



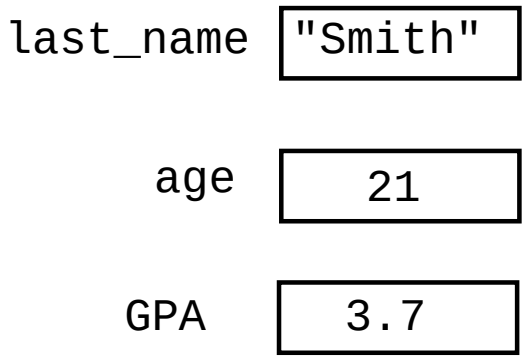
Variables



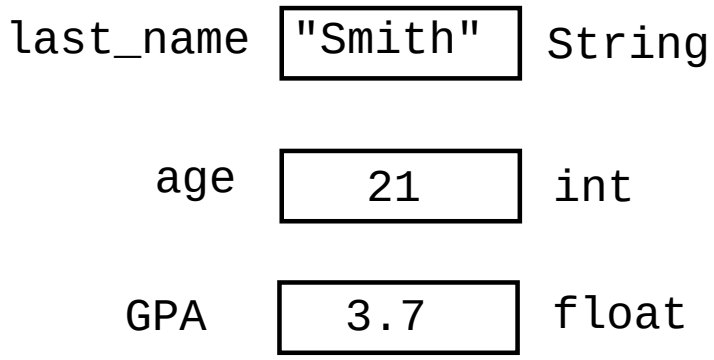
Variables



Variables



Variables



Variable declaration

- A *variable declaration* is a statement that declares tha a variable is going to be used.
- A variable declaration goes inside some method
- A variable declaration has the form:

  type identifier ;

- Examples:

```
String last_name;
int age;
float GPA;
```

Assignment

- An *assignment* is a statement that gives a value to a variable
- An assignment goes inside some method
- An assignment has the form:

  variable = *value*;

- Its meaning it to put the value into the memory location of the variable
- Examples:

```
last_name = "Smith";
age = 20;
```

- Note that the following are *incorrect*:

```
20 = age;
"Smith" = last_name;
```

Assignment

- The variable must be declared before being assigned a value

```
String last_name;
last_name = “Smith”;
```

- But the following is wrong:

```
age = 20;
int age;
```

- The type of the value must be the same as the type of the variable

```
last_name = 20; // Incorrect
age = “Smith”; // Incorrect
```

Variables and String expressions

- Variables can be used with concatenation in String expressions

  `“your age is ”+age`

- is equivalent to

  `“your age is 19”`

- if the variable age contains the value 19

A simple program

```
public class PrintData
{
  public static void main(String[] args)
  {
    String last_name;
    int age;
    last_name = “Smith”;
    age = 20;
    System.out.println(“Your last name is ” + last_name);
    System.out.println(“You are ” + age + “ years old”);
  }
}
```