Computer Architecture

- Program instructions:
  - Instructions are numbers (ultimately in binary form)
    - 00110101 represents ADD (adding numbers)
    - 10101100 represents MULT (multiplication)
    - 01010111 represents LOAD (load data from memory to a register)
    - 10100111 represents STORE (stores data from a register to memory)

Computer Architecture

- Instructions, or operators may have parameters
  - Adding the contents of registers R1 and R2 and put the result in R3:

$$\underbrace{00110101}_{ADD}\ \underbrace{10001001}_{R1}\ \underbrace{10001010}_{R2}\ \underbrace{10001011}_{R3}$$

    * Loading data from memory cell 26 and put it in register 2

$$\underbrace{11111001}_{LOAD}\ \underbrace{00011010}_{26}\ \underbrace{10001010}_{R2}$$

Computer Architecture

- Different kinds of processors have different *instruction sets* (e.g. Pentium, PowerPC, Alpha, SPARC, Motorola)
  - Each instruction set has different instructions, and associates different numbers to each type of instruction
  - Hence, a program for one type of processor cannot be directly executed by a different processor.

- Portability: the ability to run (execute) a program in more than one type of processor.

Programming Languages

- A program as understood by the computer is a long sequence of words (bits):

```
1101100011101000100100010010100101001001001010
```

  – Machine Language

- But each instruction can be written in a fashion readable by humans:

```
LOAD [26], R1
LOAD 3, R2
ADD R1, R2, R3
STORE R3, [1700000029]
```

  – Assembly language

- Assembler: a program that translates an assembly language program into its machine language equivalent.

Programming Languages

- Assembly is a low-level language
- High-level languages abstract the components of the machine

  ```
  x = y + 3;
  ```

  – Java, C, C++, Python, Perl, ML, Scheme, Prolog, Ada, Pascal, Basic, Fortran, Cobol, ...
- Abstracting the components is good: when implementing an algorithm you don't have to think about the component of the computer. You focus on the problem.
- Compiler: a program that translates a high-level language program into its machine language equivalent.

A simple Java program

```
// This is a very, very, simple program

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

A simple java program

- Java is case-sensitive:

  `HelloWorld`

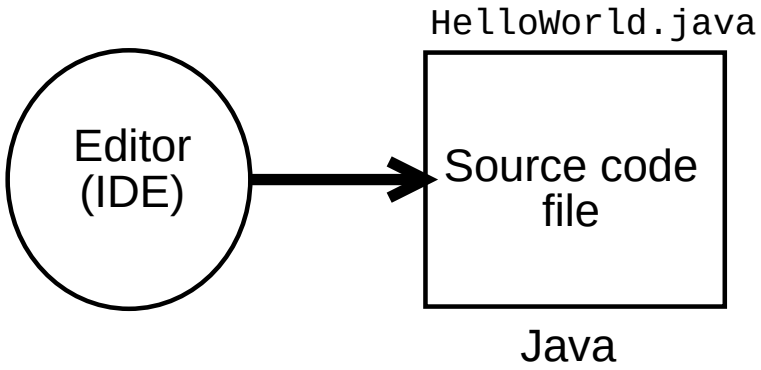
is not the same as

  `helloworld`

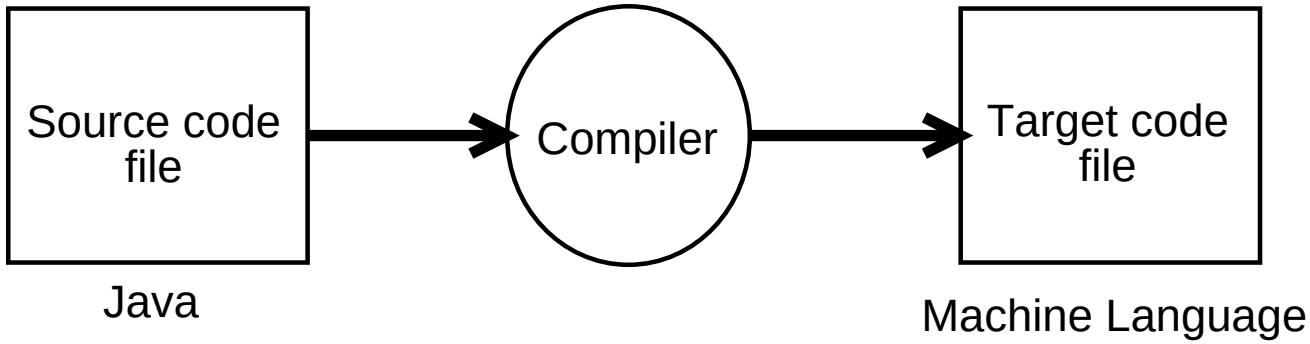
From code to a running program

- Editing
- Compilation/Interpretation
  - Compilation:
    - Translation
    - Execution
  - Interpretation:
    - Execution

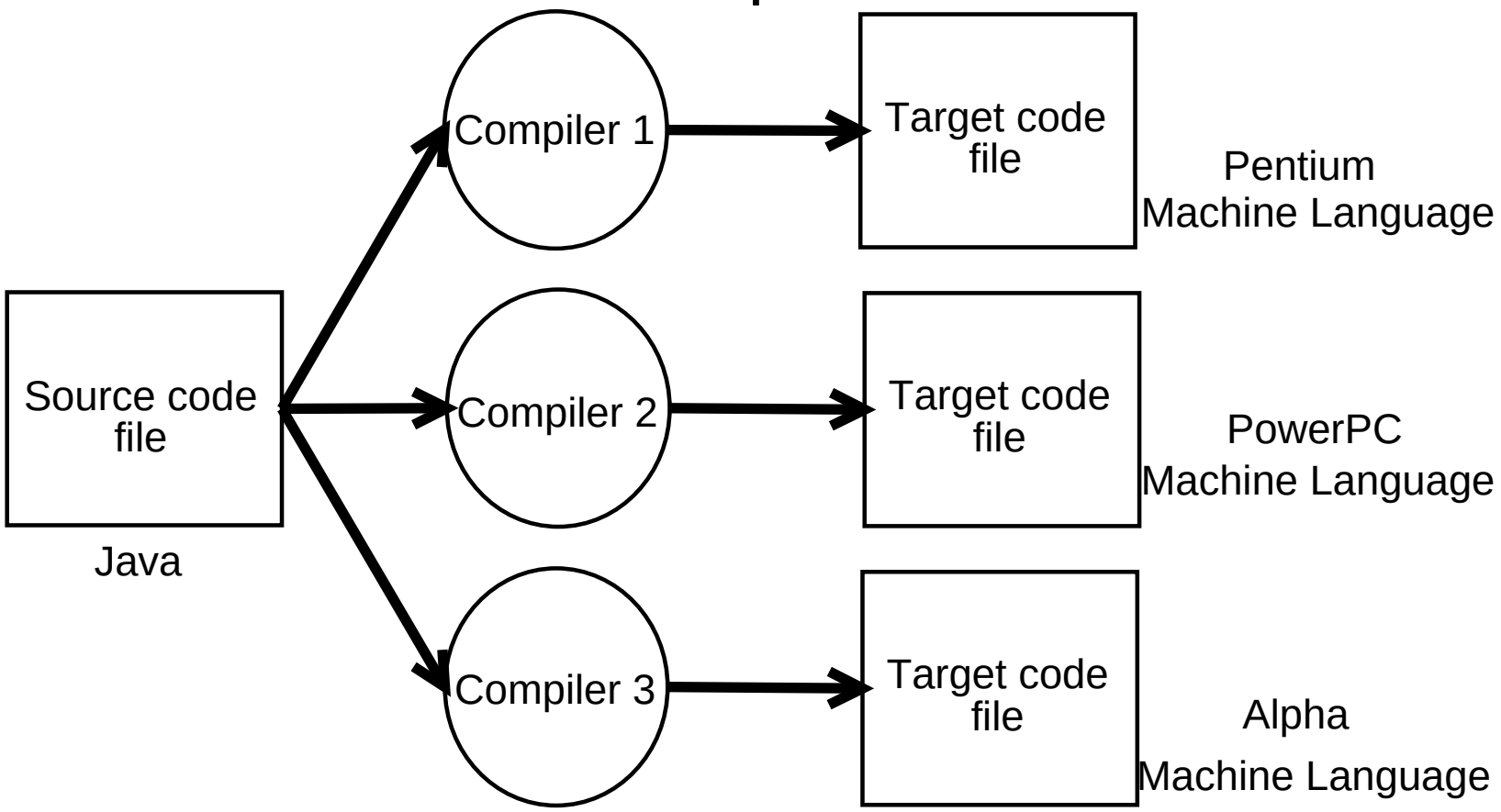
Editing



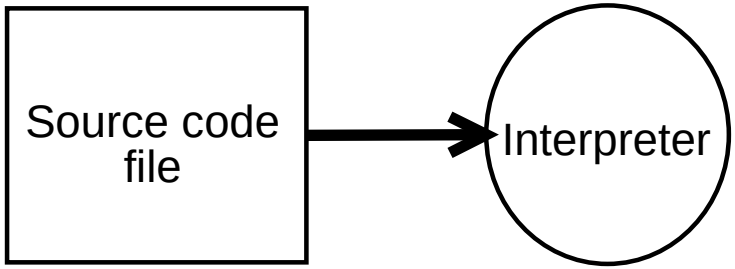
Compilers


Source code file
Compiler
Target code file
Java
Machine Language

Compilers

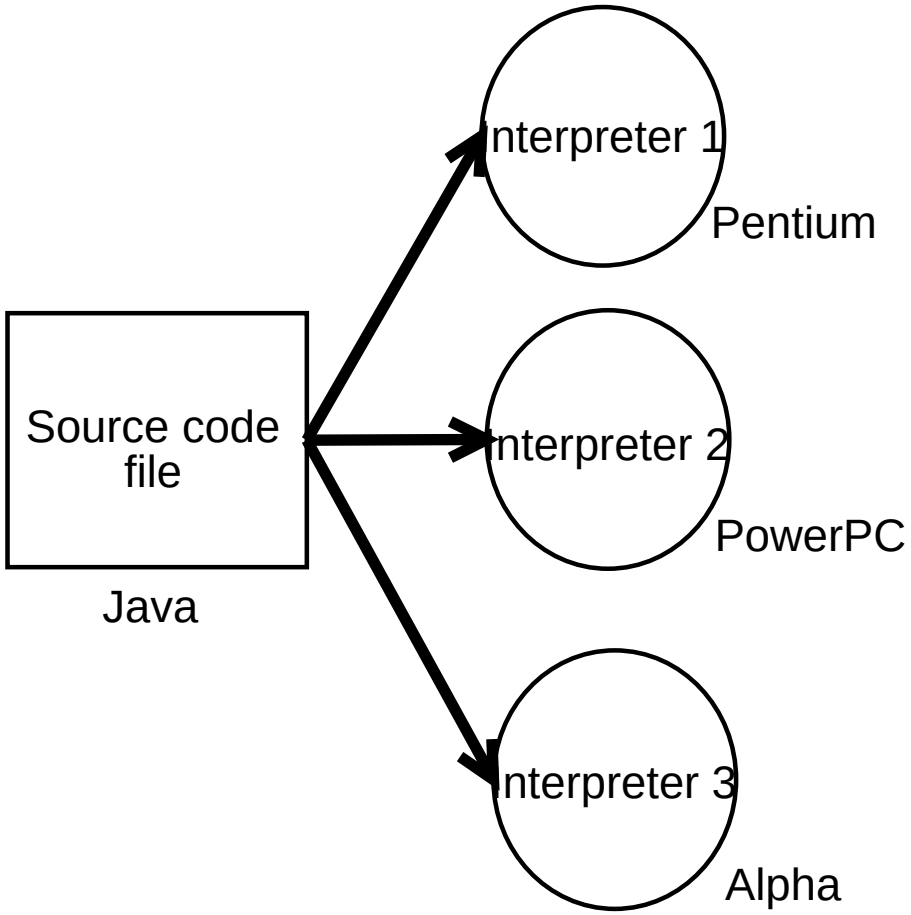

Compiler 1
Target code file
Pentium Machine Language
Source code file
Java
Compiler 2
Target code file
PowerPC Machine Language
Compiler 3
Target code file
Alpha Machine Language

Compilers

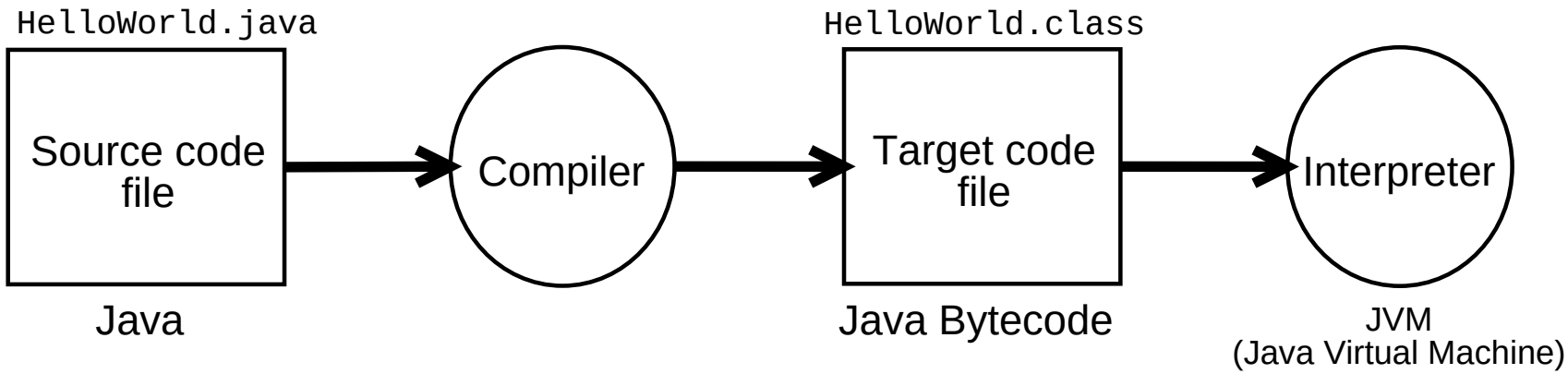


Java

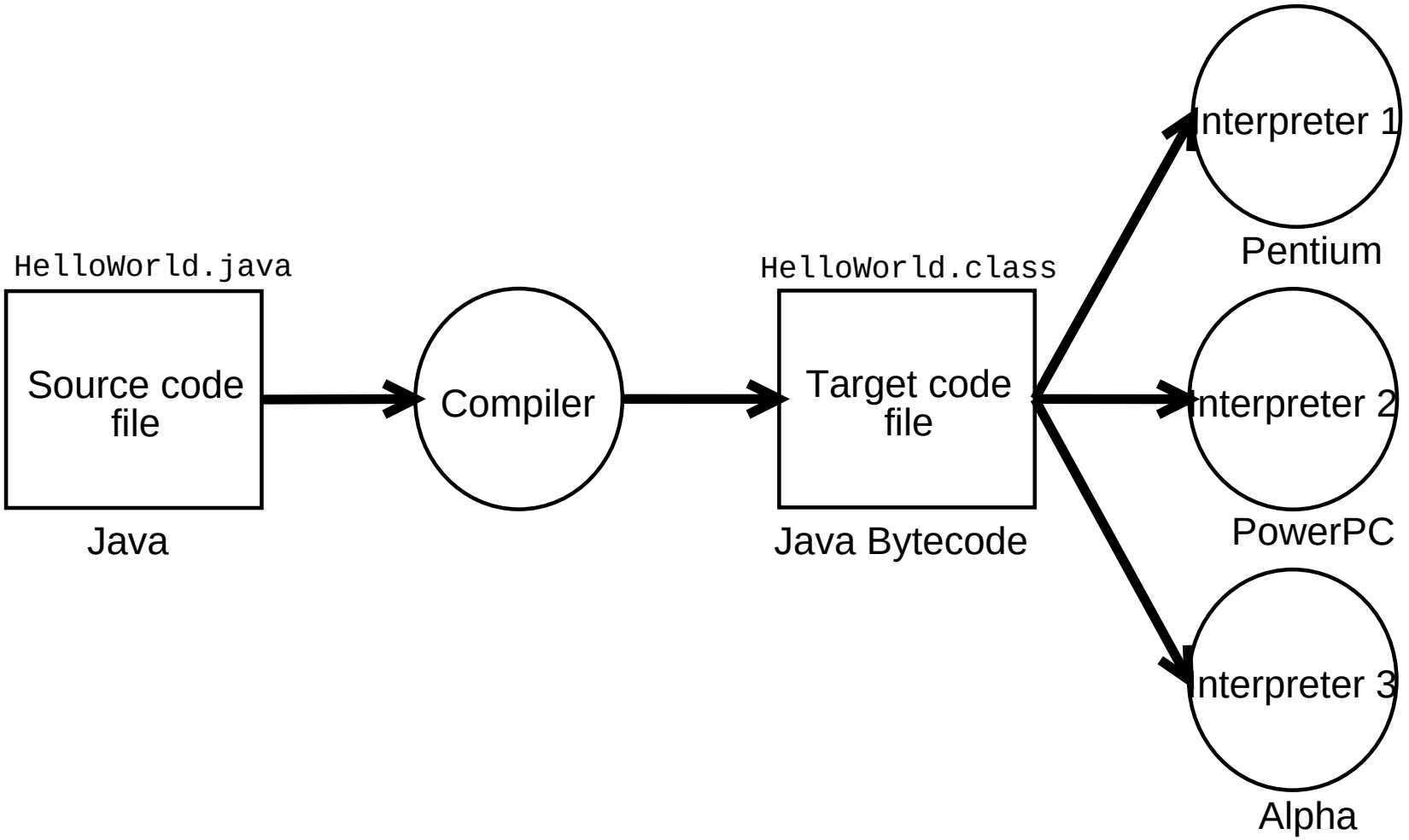
Compilers


Interpreter 1
Pentium
Source code file
Java
Interpreter 2
PowerPC
Interpreter 3
Alpha

Compilers


HelloWorld.java
Source code file
Compiler
HelloWorld.class
Target code file
Interpreter
Java
Java Bytecode
JVM
(Java Virtual Machine)

Compilers


HelloWorld.java
Source code file
Java
Compiler
HelloWorld.class
Target code file
Java Bytecode
Interpreter 1
Pentium
Interpreter 2
PowerPC
Interpreter 3
Alpha

Programming Languages

- A programming language is a formal language to describe algorithms
- A language is a means of communication
- A programming language is a means of communication between a human and a computer, but also between humans
- A programming language is formal: well-defined

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

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

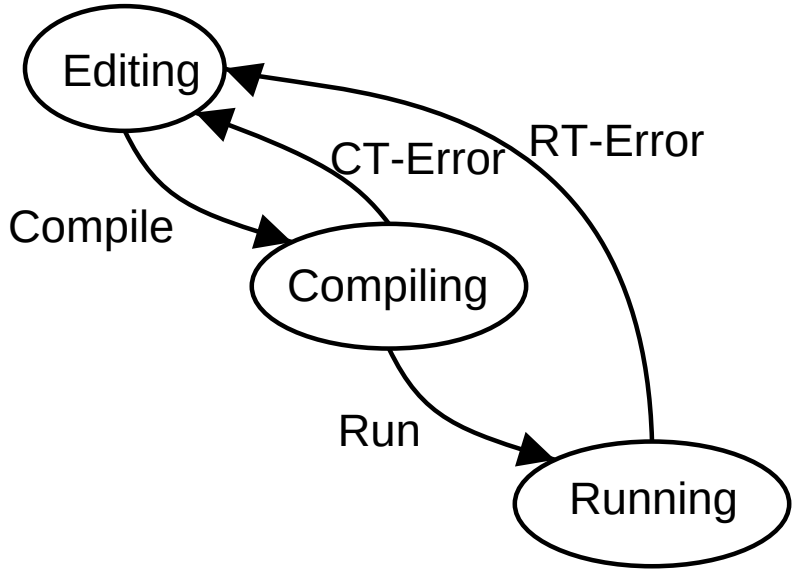
Executing programs

- 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.)

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");
    }
}
```

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

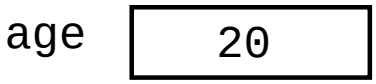
Simple programs

```
// 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.”);
    }
}
```

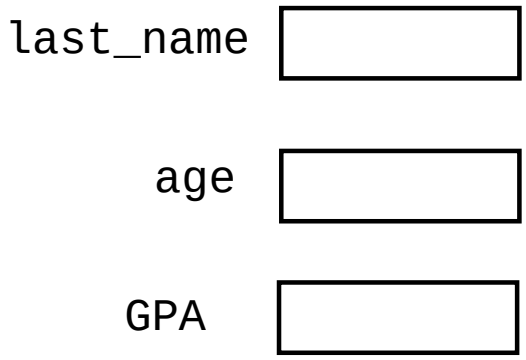
Variables

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

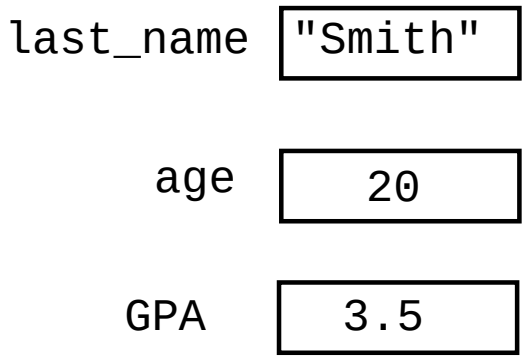
Variables



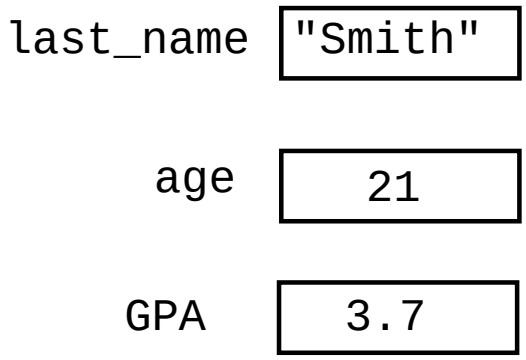
Variables



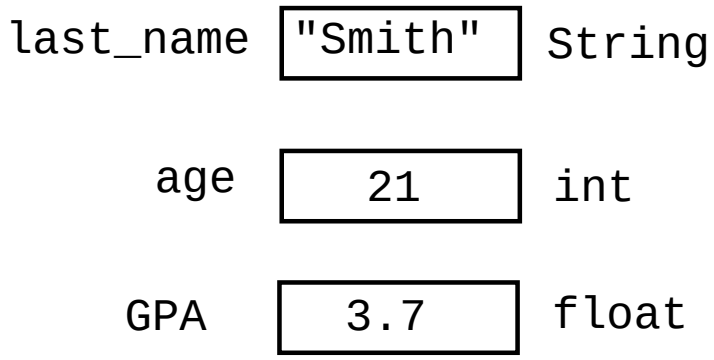
Variables



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”);
  }
}
```

Java Syntax

- Class definitions

```
public class {
    // methods
}
```

- Method definitions (inside a class)

```
// method header/signature
{
    // statements
}
```

Basic java programs

```
public class ClassName
{
    public static void main(String[] args)
    {
        // Statements
    }
}
```

Statements

- Print statement

```
System.out.println(string_expression);
```

- Variable declaration

```
type identifier;
```

- Assignment

```
variable = value;
```

- Statements in a method are executed in *sequential order* from top to bottom

Assignment

- In an assignment

```
variable = value;
```

- the variable must have been declared before,

```
x = 7; // incorrect
int x;
```

- the type of the variable must match the type of the value

```
int x;
x = "7"; // incorrect
```

Sequential execution

```
public class OrderTest
{
    public static void main(String[] args)
    {
        int a;
        int b;
        a = 2;
        b = 3;
        b = 5;
        a = 8;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Sequential execution

```
public class OrderTest
{
    public static void main(String[] args)
    {
        int a;
        int b;
        b = 5;
        a = 8;
        a = 2;
        b = 3;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Sequential execution

```
public class OrderTest
{
    public static void main(String[] args)
    {
        int a;
        int b;
        a = 2;
        b = 5;
        a = 8;
        b = 3;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Some syntactic shortcuts

- Several variables of the same type can be declared in the same variable declaration:

  type var1, var2, ..., varn ;

- Examples:

```
int a;
int b;
```

is equivalent to

```
int a, b;
```

Some syntactic shortcuts

- A variable can be initialized when declared

```
int a;
a = 2;
```

is equivalent to

```
int a = 2;
```

- But a variable cannot be redeclared, so

```
int b = 3;
int b = 2;
```

is incorrect, while the following is correct

```
int b = 3;
b = 2;
```

User Interface

- Interaction between the user and some program
- Textual UI
  - Output:

    ```
    System.out.println(string_expression);
    ```

  - Input:

    ```
    scanner.nextInt();
    scanner.nextLine();
    ```

- Examples:

```
Scanner myScanner = new Scanner(System.in);
int n;
n = myScanner.nextInt();
```

User Interface

```
import java.util.Scanner;
public class UserInputTest {
    public static void main(String[] args)
    {
        Scanner myScanner = new Scanner(System.in);
        String name;
        int age;
        System.out.print(“Enter your name: ”);
        name = myScanner.nextLine();
        System.out.print(“Enter your age: ”);
        age = myScanner.nextInt();
        System.out.println(“Your name is ” + name);
        System.out.println(“You are ” + age + “ years old”);
    }
```

```
}
```

The end