CS 149

Professor: Kevin Molloy

(adapted from slides originally developed by Alvin Chao)

Anatomy of a Java Program: Comments

Javadoc comments:

```
/**
 * Application that converts inches to centimeters.
 *
 * @author Chris Mayfield
 * @version 01/21/2014
 */
```

Everything between /** and */ ignored by compiler

Used to generate code documentation

Anatomy of a Java Program: Comments

Block comments are used for text that should *not* be part of the published documentation:

```
/*
   Permission is hereby granted, free of charge, to any
   person obtaining a copy of this software and associated
   documentation files (the "Software"), to deal in the
   Software without restriction.
*/
```

In-line comments are used for short clarifying statements:

```
// Create a scanner for standard input.
```

Anatomy of a Java Program: Classes

Java is an object-oriented language (OO)

Java classes tie together instructions and data

All Java code ***must*** exist within some class

```
public class ConvertInches {

}
```

`public` and `class` are keywords: Words that have a special meaning for Java.

`public` – (more later)

`class` – Create a class with the following name. (Must match the file name)

Class names are always capitalized (by convention)

Braces { and } enclose blocks of code

Anatomy of a Java Program: Methods

Method – named collection of Java statements:

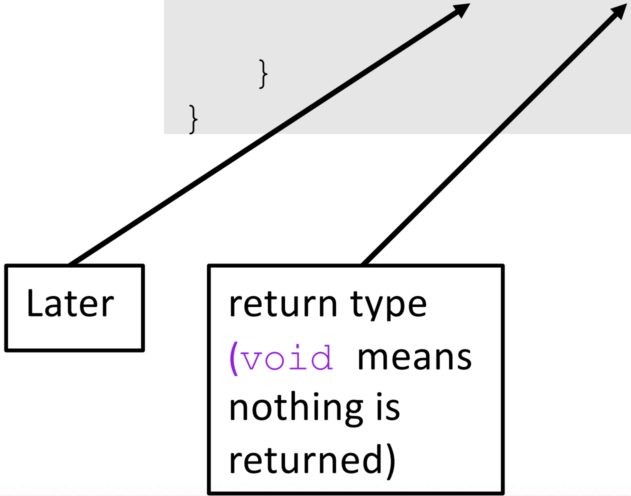
```
public class ConvertInches {

    public static void main(String[] args) {

    }
}
```

Later

Anatomy of a Java Program: Methods

Method – named collection of Java statements:

```
public class ConvertInches {

   public static void main(String[] args) {

   }
}
```

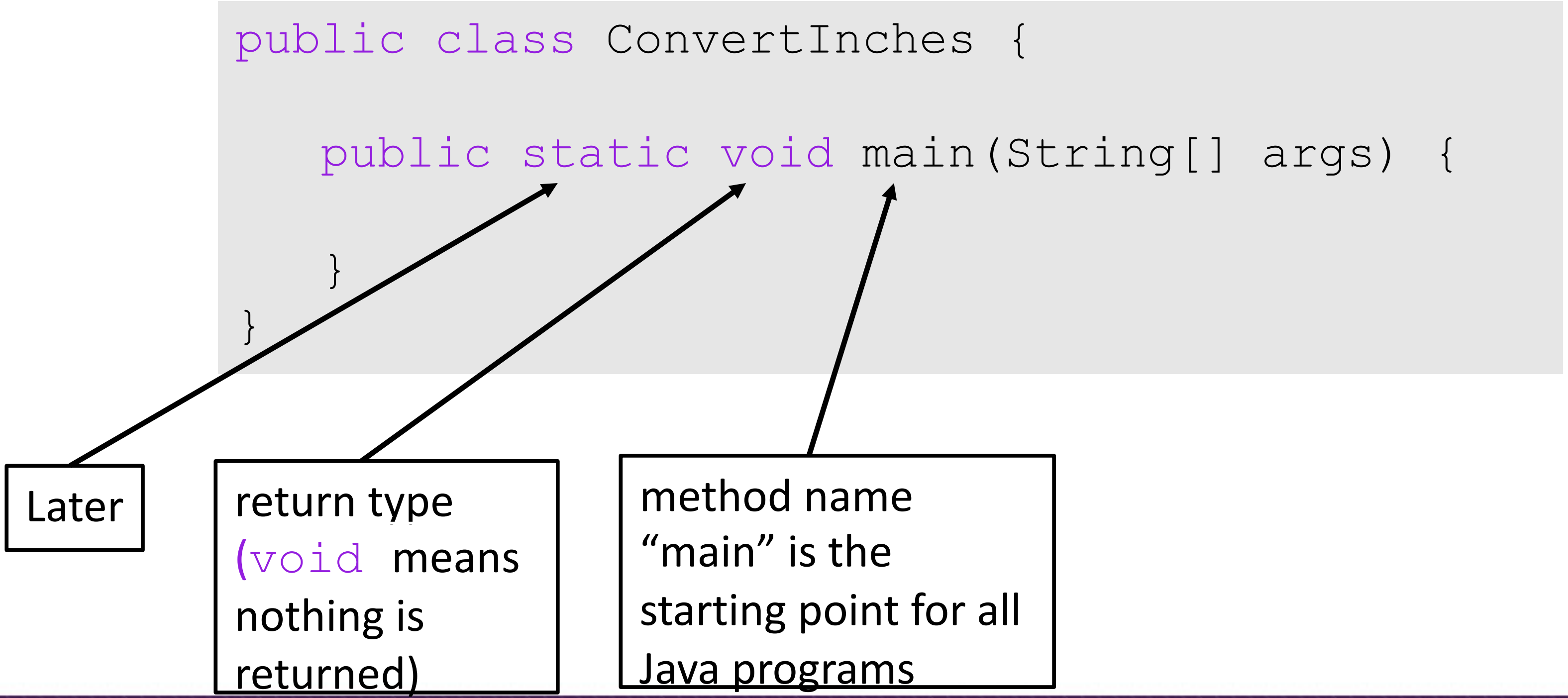


Later

return type (void means nothing is returned)

Anatomy of a Java Program: Methods

Method – named collection of Java statements:



```
public class ConvertInches {

   public static void main(String[] args) {

   }
}
```

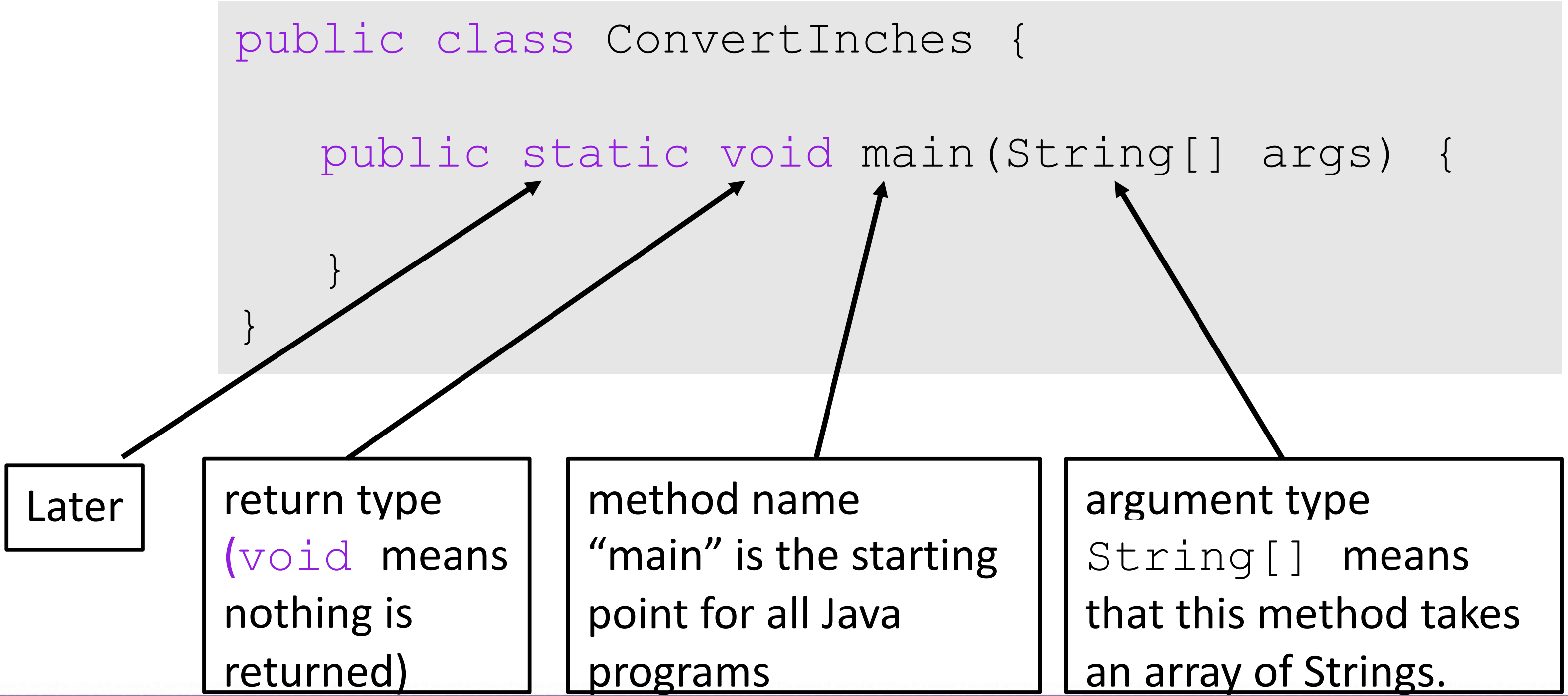
Later

return type (void means nothing is returned)

method name "main" is the starting point for all Java programs

Anatomy of a Java Program: Methods

Method – named collection of Java statements:



```
public class ConvertInches {

   public static void main(String[] args) {

   }
}
```

Later

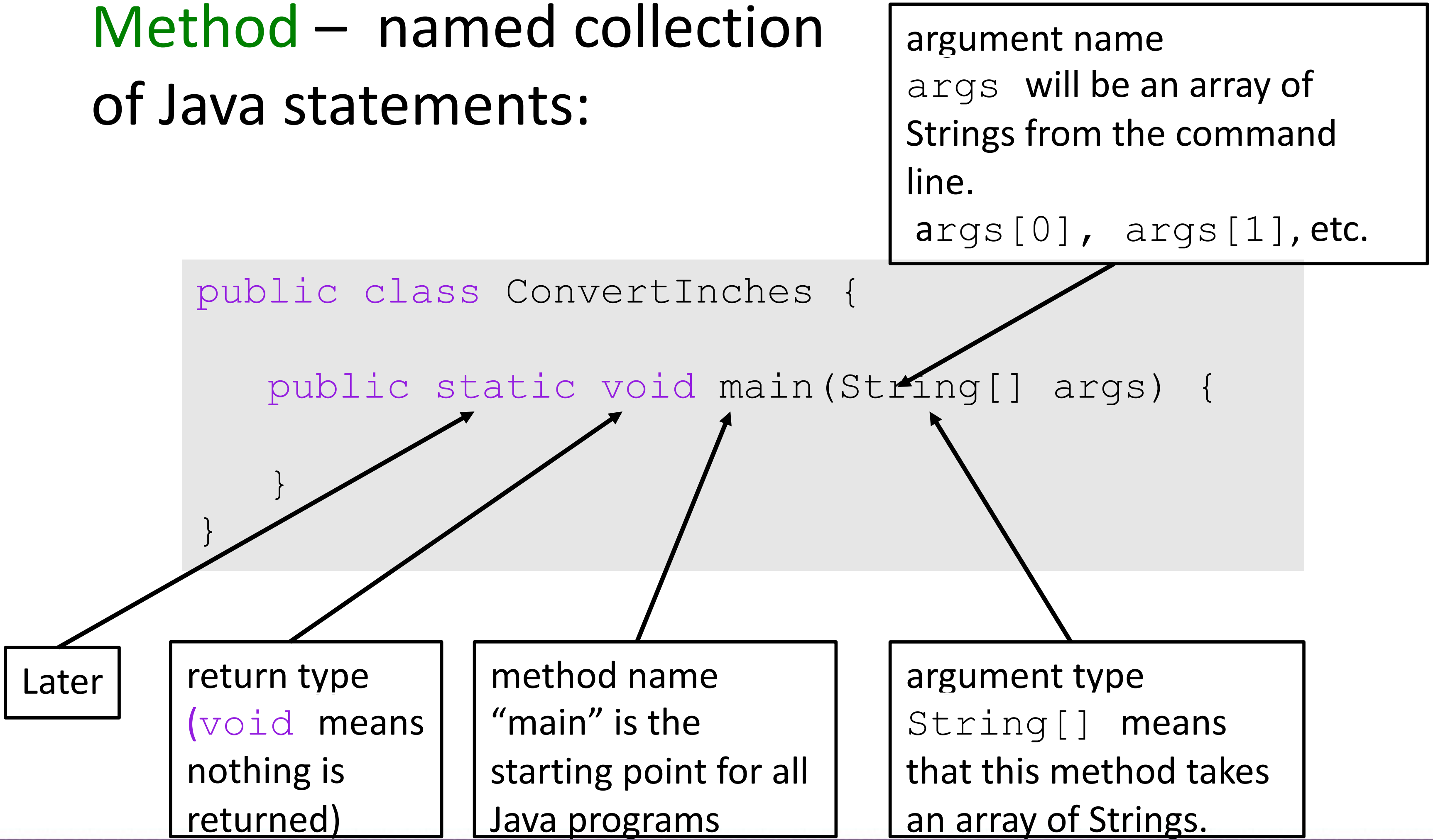
return type (void means nothing is returned)

method name "main" is the starting point for all Java programs

argument type String[] means that this method takes an array of Strings.

Anatomy of a Java Program: Methods

Method – named collection of Java statements:



argument name `args` will be an array of Strings from the command line. `args[0], args[1],` etc.

```
public class ConvertInches {

   public static void main(String[] args) {

   }
}
```

Later

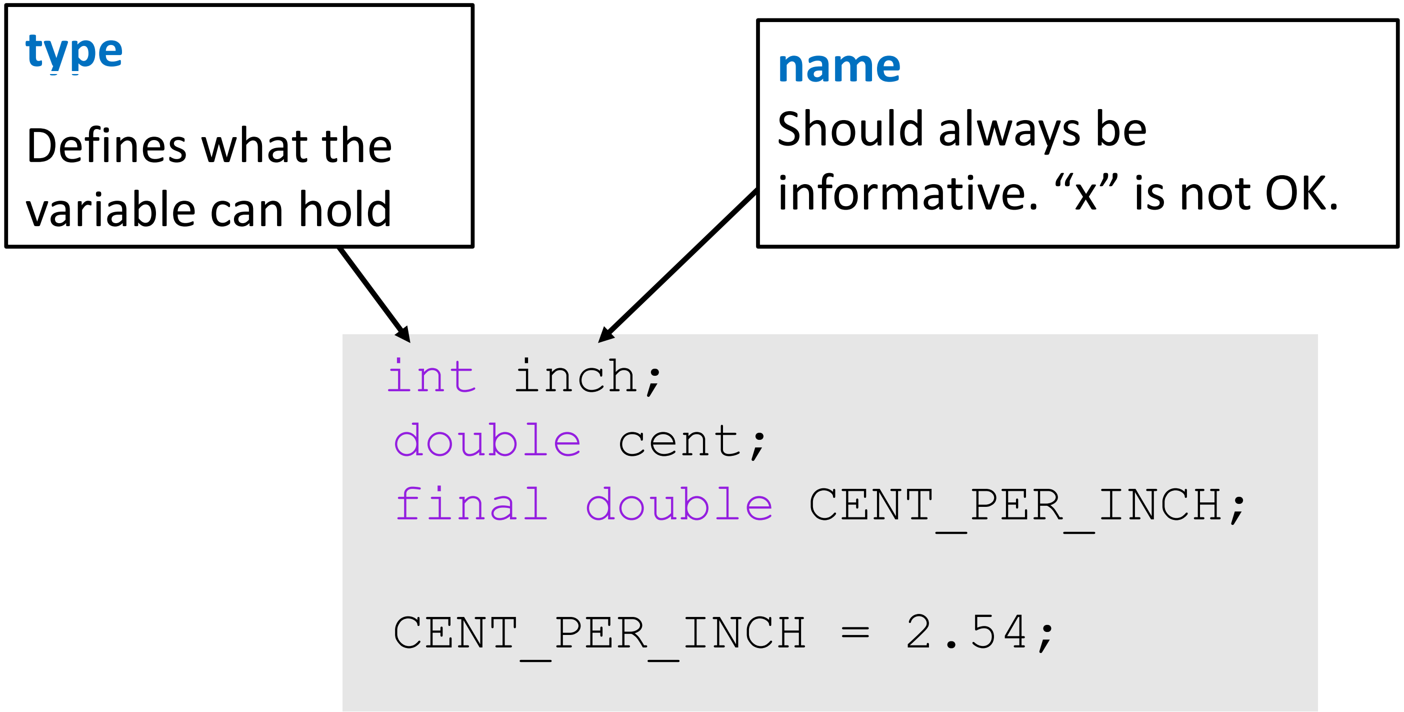
return type (`void` means nothing is returned)

method name "main" is the starting point for all Java programs

argument type `String[]` means that this method takes an array of Strings.

Anatomy of a Java Program: Declaring and Assigning Variables

variable – named box for storing data:



type

Defines what the variable can hold

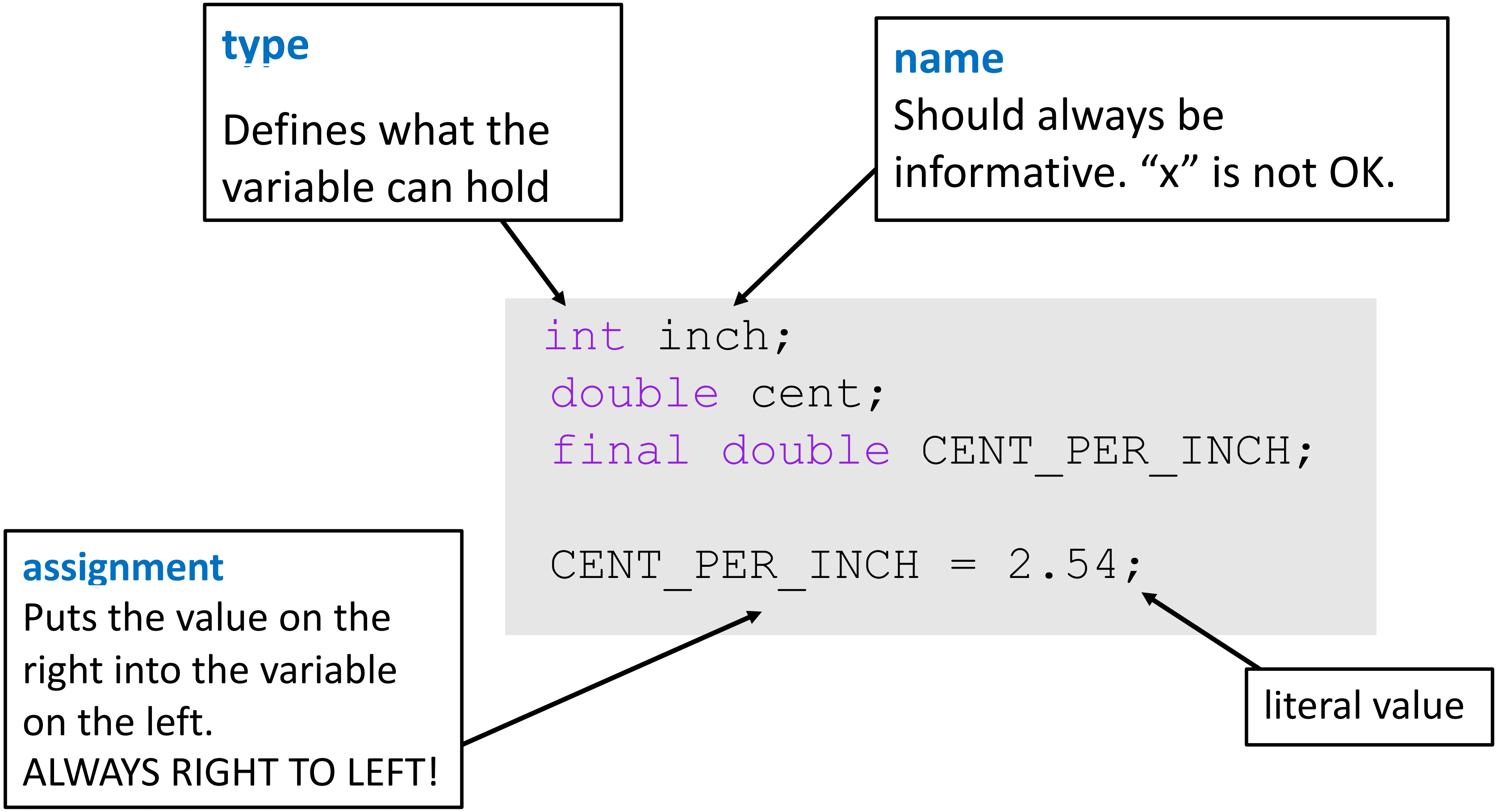
name

Should always be informative. “x” is not OK.

```
int inch;
double cent;
final double CENT_PER_INCH;

CENT_PER_INCH = 2.54;
```

Anatomy of a Java Program: Declaring and Assigning Variables

variable – named box for storing data:



type

Defines what the variable can hold

name

Should always be informative. “x” is not OK.

```
int inch;
double cent;
final double CENT_PER_INCH;

CENT_PER_INCH = 2.54;
```

assignment

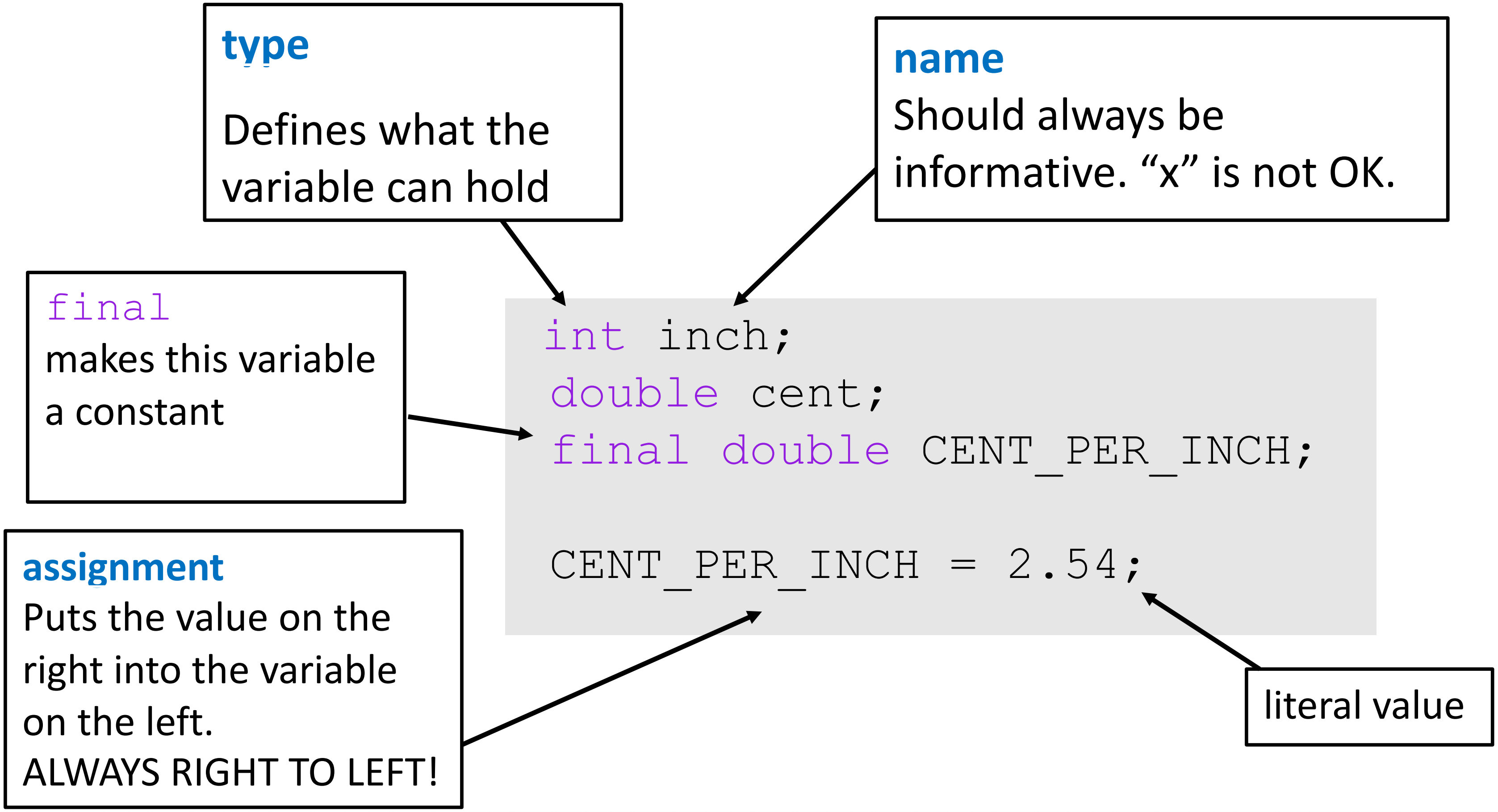
Puts the value on the right into the variable on the left.

ALWAYS RIGHT TO LEFT!

literal value

Anatomy of a Java Program: Declaring and Assigning Variables

variable – named box for storing data:



type

Defines what the variable can hold

name

Should always be informative. “x” is not OK.

final

makes this variable a constant

```
int inch;
double cent;
final double CENT_PER_INCH;

CENT_PER_INCH = 2.54;
```

assignment

Puts the value on the right into the variable on the left.

ALWAYS RIGHT TO LEFT!

literal value

Anatomy of a Java Program: Standard Library and Keyboard Input

```
import java.util.Scanner;

/**
* Application that converts inches to centimeters.
*
* @author Chris Mayfield
* @version 01/21/2014
*/
public class ConvertInches {

   public static void main(String[] args) {
      int inch;
      double cent;
      final double CENT_PER_INCH;
      CENT_PER_INCH = 2.54;

      // Create a scanner for standard input.
      Scanner keyboard;
      keyboard = new Scanner(System.in);

      // Prompt the user and get the value.
      System.out.print("How many inches? ");
      inch = keyboard.nextInt();
```

import
“Brings in”
external classes

The Scanner class, along with System.in are used to read user input from the terminal

Putting it all together...

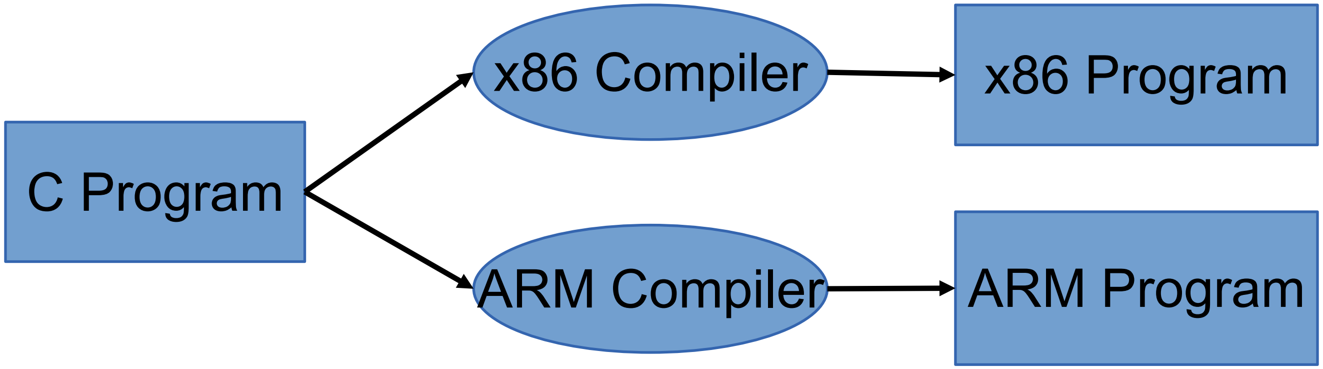
```
public class ConvertInches {

   public static void main(String[] args) {
      int inch;
      double cent;
      final double CENT_PER_INCH;
      CENT_PER_INCH = 2.54;

      // Create a scanner for standard input.
      Scanner keyboard;
      keyboard = new Scanner(System.in);

      // Prompt the user and get the value.
      System.out.print("How many inches? ");
      inch = keyboard.nextInt();

      // Convert and output the result.
      cent = inch * CENT_PER_INCH;
      System.out.print(inch + "in = ");
      System.out.println(cent + "cm ");
   }

}
```

multiplication

+ joins strings (or adds numbers)

Reminder: Portability

Most “high-level” languages are considered portable because they can be compiled into machine code for any computer:



C Program

x86 Compiler

ARM Compiler

x86 Program

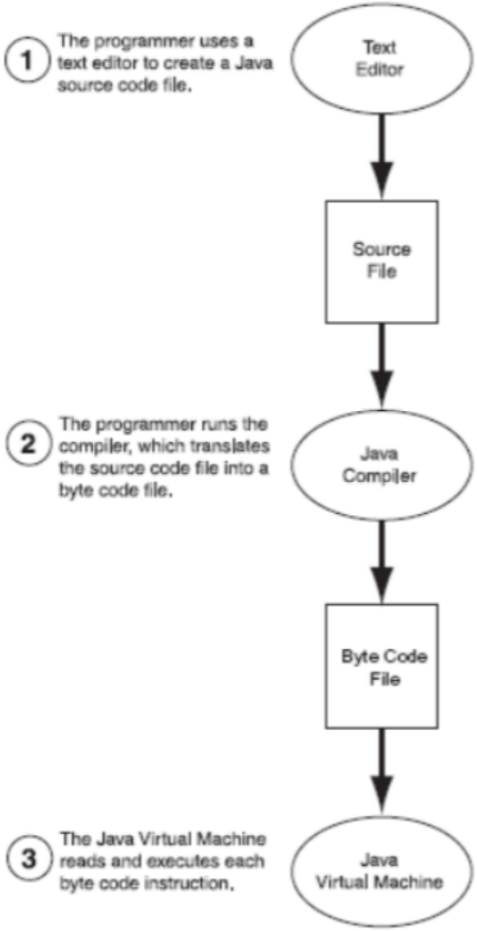
ARM Program

Java Compilation

Byte Code Files are portable because there are JVM's that run on most machines

The **same** compiled byte code works on any JVM

Figure 1-5
Program development process



1 The programmer uses a text editor to create a Java source code file.

2 The programmer runs the compiler, which translates the source code file into a byte code file.

3 The Java Virtual Machine reads and executes each byte code instruction.

Which is Syntactically Correct?

```
public static void main(String[] args)
{
      System.out.println("Hello " + args[0] + "!");
      System.out.println("Welcome to CS149.");
}
```

```
public class Personal {
   public static void main(String[] args)
   {
      System.out.println("Hello " + args[0] + "!");
      System.out.println("Welcome to CS149.");
   }
}
```

```
public class Personal
{
   // public static void main(String[] args)
   {
      System.out.println("Hello " + args[0] + "!");
      System.out.println("Welcome to CS149.");
   }
}
```

Which is Syntactically Correct? (File name is Good.java)

```
public class Welcome {
   public static void main(String[] args)
   {
      String name;
      name = "Bob";
      System.out.println("Hello " + name + "!");
      System.out.println("Welcome to CS149.");
   }
}
```

```
public class Good {
   public static void main(String[] args)
   {
      String name;
      "Bob" = name;
      System.out.println("Hello " + name + "!");
      System.out.println("Welcome to CS149.");
   }
}
```

```
public class Good {
   public static void main(String[] args)
   {
      String name;
      name = "Bob";
      System.out.println("Hello " + name + "!");
      System.out.println("Welcome to CS149.");
   }
}
```

Which is Syntactically Correct?

```
public class Good
   public static void main(String[] args)
   {
      String name;
      name = "Bob";
      System.out.println("Hello " + name + "!");
      System.out.println("Welcome to CS149.");
   }
}
```

```
public class Good {
   public static void main(String[] args)
   {
      String name;
      name = "Bob";
      System.out.println("Hello " + name + "!")
      System.out.println("Welcome to CS149.");
   }
}
```

```
public class Good {
   public static void main(String[] args){
      String name; name = "Bob";
        System.out.println("Hello " + name + "!");
     System.out.println("Welcome to CS149.");}
}
```