Lecture 1

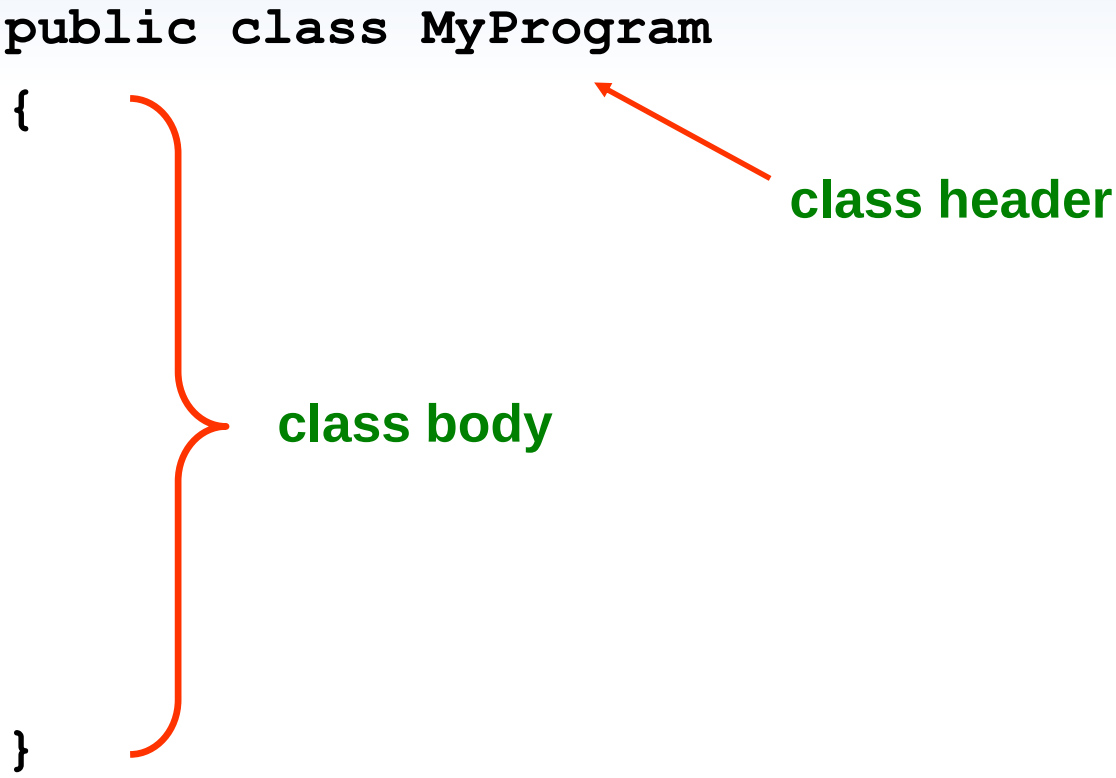
Learning Objectives

- Introduction to java programming language
- Declaring Java primitive type variables
- Variable Initialization
- character strings and String objects
- Print and println methods
- String Concatenation
- Escape sequences

Java

- A programming language specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid program statements
- In the Java programming language
  - a program is made up of one or more *classes*
  - a class contains one or more *methods*
  - a method contains program *statements*
- A Java application always contains a method called `main`

A Java Program



```
public class MyProgram
{

   public static void main(String[] args)
   {
      System.out.println(“Welcome to Fundamentals Computer programming");

   }
}
```

A Java Program

```
//  comments about the class
public class MyProgram
{
   //  comments about the method
   public static void main(String[] args)
   {

   }

}
```

method body

method header

Comments

- Comments should be included to explain the purpose of the program and describe processing
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```

Reserved Words

- Are special identifiers that already have a predefined meaning in the language
- Java reserved words:

abstract	default	goto*	package	this
assert	do	if	private	throw
boolean	double	implements	protected	throws
break	else	import	public	transient
byte	enum	instanceof	return	true
case	extends	int	short	try
catch	false	interface	static	void
char	final	long	strictfp	volatile
class	finally	native	super	while
const*	float	new	switch	
continue	for	null	synchronized	

Identifiers

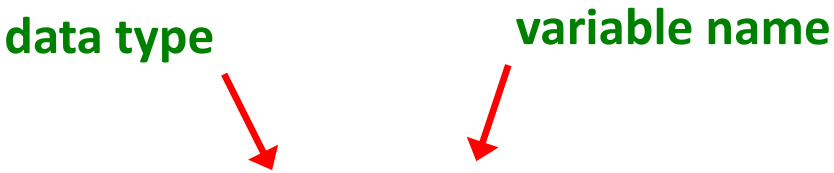
- *Identifiers* are the words a programmer uses in a program
  - can be made up of letters, digits, the underscore character (_), and the dollar sign
  - **cannot begin with a digit**
- Java is *case sensitive*
  - `Total`, `total`, and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers, such as
  - *title case* for class names - `Lincoln`
  - *upper case* for constants - `MAXIMUM`

White Space

- Spaces, blank lines, and tabs are called *white space*
- **White space is used to separate words and symbols in a program**
- Extra white space is ignored
- A valid Java program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying its name and the type of information that it will hold




```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Primitive Data Types

- There are eight primitive data types in Java
- Four of them represent integers
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers
  - `float`, `double`
- And one of them represents boolean values
  - `boolean`
- One of them represents characters
  - `char`

Numeric Types

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

Type	Storage	Min Value	Max Value
`byte`	8 bits	–128	127
`short`	16 bits	–32,768	32,767
`int`	32 bits	–2,147,483,648	2,147,483,647
`long`	64 bits	–9,223,372,036,854,775,808	9,223,372,036,854,775,807
`float`	32 bits	Approximately –3.4E+38 with 7 significant digits	Approximately 3.4E+38 with 7 significant digits
`double`	64 bits	Approximately –1.7E+308 with 15 significant digits	Approximately 1.7E+308 with 15 significant digits

Booleans

- A `boolean` value represents a true or false condition
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done;
```

- A `boolean` variable can also be used to represent any two states, such as a light bulb being on or off

Characters

- A `char` variable stores a single character
- Character literals are delimited by single quotes:

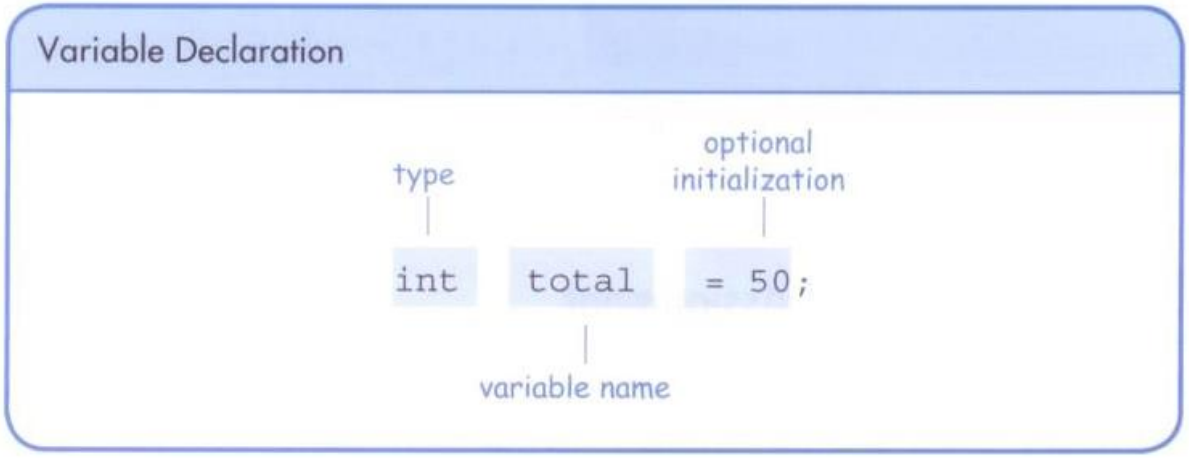
```
'a'    'X'    '7'    '$'    ','    '\n'
```

- Example declarations

```
char topGrade;

char terminator;
```

Variable Initialization

- A variable can be given an initial value in the declaration



- When a variable is used in a program, its current value is used

```
//********************************************************************
//  PianoKeys.java       Java Foundations
//
//  Demonstrates the declaration, initialization, and use of an
//  integer variable.
//********************************************************************

public class PianoKeys
{
   //-----------------------------------------------------------------
   //  Prints the number of keys on a piano.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      int keys = 88;

      System.out.println("A piano has " + keys + " keys.");
   }
}
```

```
A piano has 88 keys.
```

The String Class

- A primitive character variable can only hold one character, if we want to store more than one character in a variable, we need to declare a `String` object, which can hold multiple characters.
- The **String** class can be used to declare a string *object reference variable*

Creating String Objects

- Generally, we use the `new` operator to create an object:

```
String title = new String("James Gosling");
```

This calls the String *constructor*, which is a special method that sets up the object

- Creating an object is called *instantiation*
- An object is an *instance* of a particular class

Creating String objects

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
String title;

title = "Java rocks!";
```

- This is special syntax that works only for strings
- Each string literal (enclosed in double quotes) represents a `String` object

Constants

- A *constant* is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- In Java, we use the **`final`** modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Constants

- Constants are useful for three important reasons
  - First, they give meaning to otherwise unclear literal values
    - For example, `MAX_LOAD` means more than the literal 250
  - Second, they facilitate program maintenance
    - If a constant is used in multiple places, its value need only be updated in one place
  - Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

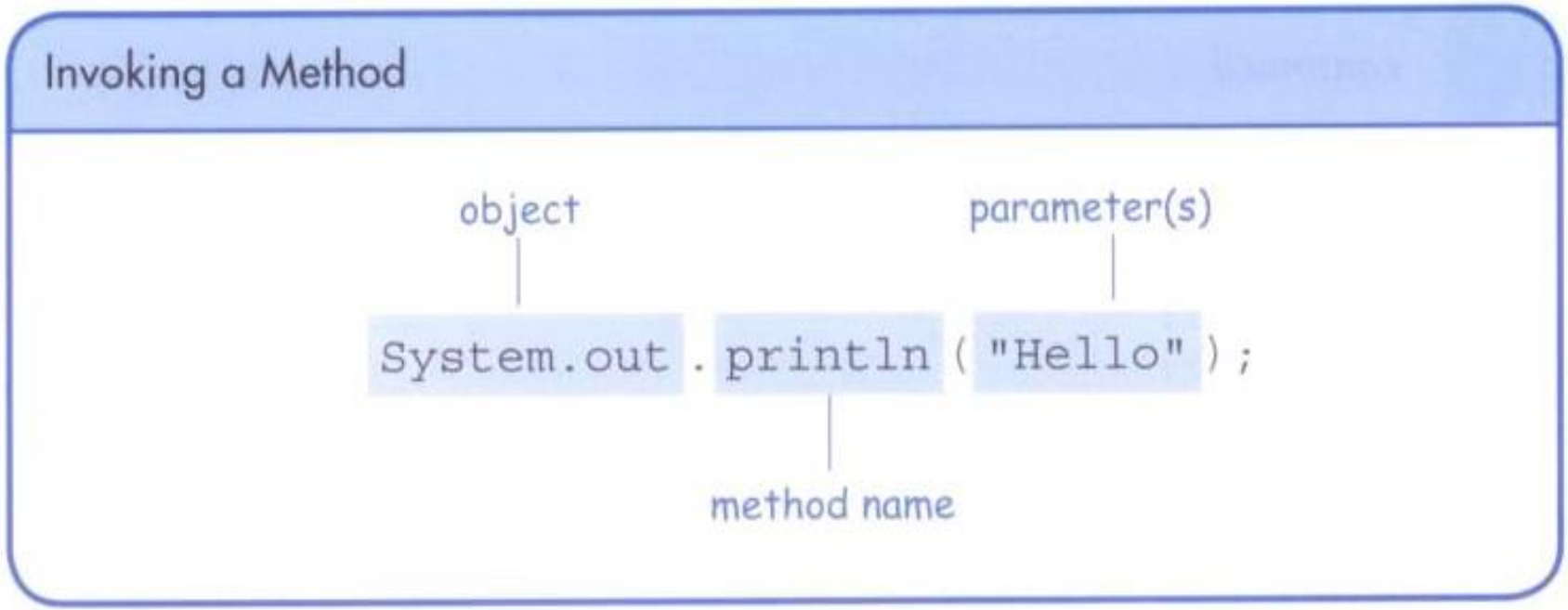
Examples of Declaring Java Primitive Type Variables


```
int total;
```

```
float price;

double totalcost;

char topGrade;

String title = new String(" Java rocks ");
```

The println Method

- The `System.out` object represents a destination (the monitor) to which we can send output



The print Method

- The `System.out` object provides another service as well
- The `print` method is similar to the `println` method, except that it does not advance to the next line
- Therefore anything printed after a `print` statement will appear on the same line

```
//********************************************************************
//  Countdown.java       Java Foundations
//
//  Demonstrates the difference between print and println.
//********************************************************************

public class Countdown
{
   //-----------------------------------------------------------------
   //  Prints two lines of output representing a rocket countdown.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.print("Three... ");
      System.out.print("Two... ");
      System.out.print("One... ");
      System.out.print("Zero... ");

      System.out.println("Liftoff!"); // appears on first output line

      System.out.println("Houston, we have a problem.");
   }
}
```

```
Three... Two... One... Zero... Liftoff!
Houston, we have a problem.
```

String Concatenation

- A string literal cannot be broken across two lines in a program
- The *string concatenation operator* (+) is used to append one string to the end of another

```
"Peanut butter " + "and jelly"
```

- It can also be used to append a number to a string

```
//********************************************************************
//  Facts.java       Java Foundations
//
//  Demonstrates the use of the string concatenation operator and the
//  automatic conversion of an integer to a string.
//********************************************************************

public class Facts
{
   //-----------------------------------------------
   //  Prints various facts.
   //-----------------------------------------------
   public static void main(String[] args)
   {
      // Strings can be concatenated into one long string
      System.out.println("We present the following facts for your "
                         + "extracurricular edification:");

      System.out.println();

      // A string can contain numeric digits
      System.out.println("Letters in the Hawaiian alphabet: 12");

      // A numeric value can be concatenated to a string
      System.out.println("Dialing code for Antarctica: " + 672);

      System.out.println("Year in which Leonardo da Vinci invented "
                         + "the parachute: " + 1515);

      System.out.println("Speed of ketchup: " + 40 + " km per year");
   }
}
```

```
We present the following facts for your extracurricular edification:

Letters in the Hawaiian alphabet: 12
Dialing code for Antarctica: 672
Year in which Leonardo da Vinci invented the parachute: 1515
Speed of ketchup: 40 km per year
```

String Concatenation

- The + operator is also used for arithmetic addition
- The function that it performs depends on the type of the information on which it operates
- **If both operands are strings, or if one is a string and one is a number, it performs string concatenation**
- If both operands are numeric, it adds them
- The + operator is evaluated left to right, but parentheses can be used to force the order

```
//********************************************************************
//  Addition.java       Java Foundations
//
//  Demonstrates the difference between the addition and string
//  concatenation operators.
//********************************************************************

public class Addition
{
   //-----------------------------------------------------------------
   //  Concatenates and adds two numbers and prints the results.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.println("24 and 45 concatenated: " + 24 + 45);

      System.out.println("24 and 45 added: " + (24 + 45));
   }
}
```

```
24 and 45 concatenated: 2445
24 and 45 added: 69
```

Escape Sequences

- What if we wanted to print a the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println("I said \"Hello\" to you.");
```

Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
\b	backspace
\t	tab
\n	newline
\r	carriage return
\"	double quote
\'	single quote
\\	backslash

```
//********************************************************************
//  Roses.java       Java Foundations
//
//  Demonstrates the use of escape sequences.
//********************************************************************

public class Roses
{
   //-----------------------------------------------------------------
   //  Prints a poem (of sorts) on multiple lines.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.println("Roses are red,\n\tViolets are blue,\n" +
         "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
         "So I'd rather just be friends\n\tAt this point in our " +
         "relationship.");
   }
}
```

```
Roses are red,
        Violets are blue,
Sugar is sweet,
        But I have "commitment issues",
        So I'd rather just be friends
        At this point in our relationship.
```