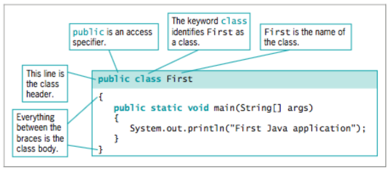
Ch. 1 – Creating Java Programs // **Syntax error**: misspelling or misuse of java keywords *won't* compile // **Semantic error**: *aka Logical error will compile* // **Attribute**: *class fields, created for each object*

Keywords:

- **Object-Oriented Programming**: creating classes, objects and applications that manipulate them
- **Encapsulation**: concealment of data a methods (scope)
- **Inheritance**: taking on the attributes and methods of a parent class
- **Polymorphism**: using the same symbols or keywords in a different manner depending on context

- **Source code**: The code you write.
- **Bytecode**: Java Compiler turns source code into byte code (1000111) read by JVM
- **Identifier**: the name you give a variable or method
- **Pascal casing**: capitalize each word with no spaces AClassName
- **Camel Casing**: capitalize each subsequent, but not the first aMethodName



```
public class First
{
   public static void main(String[] args)
   {
      System.out.println("First Java application");
   }
}
```

public is an access specifier.

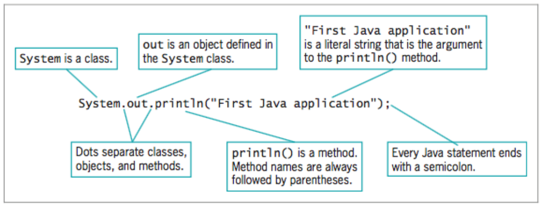
The keyword class identifies First as a class.

First is the name of the class.

This line is the class header.

Everything between the braces is the class body.

Figure 1-6 The parts of a typical class



```
System.out.println("First Java application");
```

System is a class.

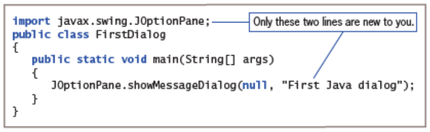
out is an object defined in the System class.

"First Java application" is a literal string that is the argument to the println() method.

Dots separate classes, objects, and methods.

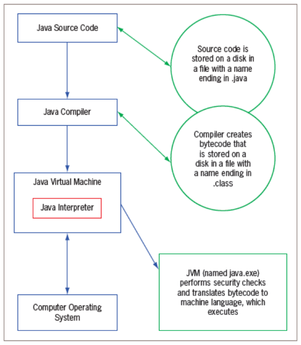
println() is a method. Method names are always followed by parentheses.

Every Java statement ends with a semicolon.



```
import javax.swing.JOptionPane;
public class FirstDialog
{
   public static void main(String[] args)
   {
      JOptionPane.showMessageDialog(null, "First Java dialog");
   }
}
```

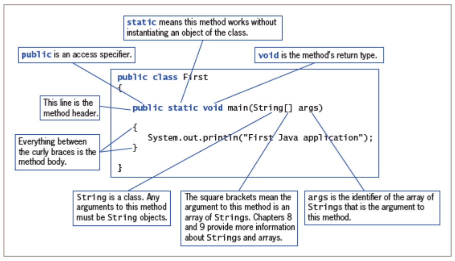
Only these two lines are new to you.



Source code is stored on a disk in a file with a name ending in .java

Compiler creates bytecode that is stored on a disk in a file with a name ending in .class

JVM (named java.exe) performs security checks and translates bytecode to machine language, which executes



```
public class First
{
   public static void main(String[] args)
   {
      System.out.println("First Java application");
   }
}
```

static means this method works without instantiating an object of the class.

public is an access specifier.

void is the method's return type.

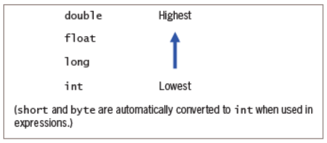
This line is the method header.

Everything between the curly braces is the method body.

String is a class. Any arguments to this method must be String objects.

The square brackets mean the argument to this method is an array of Strings. Chapters 8 and 9 provide more information about Strings and arrays.

args is the identifier of the array of Strings that is the argument to this method.



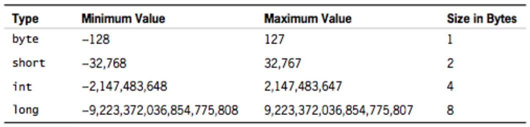
(short and byte are automatically converted to int when used in expressions.)

Figure 2-41 Order for establishing unifying data types

Ch. 2 – Using Data // **Allman style**: { } align // **JAVA IS CASE SENSITIVE!!** // **Primitive data types**: (8 total) – byte, short, int, long, float, double, char, boolean -- STRING EXCLUDED!!

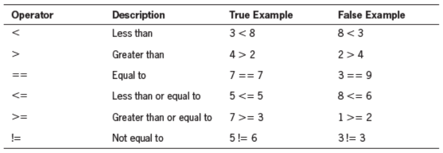
- **Char**: a single character: 'a' ; use single quotes
- **Float**: decimal value (about 4 decimal places will show), must typecast: 3.1415f;
- **Double**: decimal value (about 8 decimal places ish)
- **Parse**: turning a string into a number: Integer.parseInt or Double.parseDouble
- **Input dialog box**: GUI input for Strings: JoptionPane.showInputDialog(null, "your string")
- When a string and a numeric value are concatenated, the resulting expression is a String. Example "X" + 2 + 4 results in "X24"

- **confirm dialog box**: GUI Yes, No or Cancel: Yes = 0, No = 1, cancel = 2
- **Type conversions**: promoting to a higher precision number
- **Type Cast**: Manually forcing a data type: (byte)3 or for Float and long 3f or 3L
- **Arithmetic**: Int * Int = int , Double * Int = Double (same for division)
- **Modulus (%)**: returns the remainder only 6%4 = 2



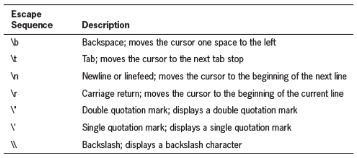
Type	Minimum Value	Maximum Value	Size in Bytes
byte	–128	127	1
short	–32,768	32,767	2
int	–2,147,483,648	2,147,483,647	4
long	–9,223,372,036,854,775,808	9,223,372,036,854,775,807	8

Table 2-2 Limits on integer values by type



Operator	Description	True Example	False Example
<	Less than	3 < 8	8 < 3
>	Greater than	4 > 2	2 > 4
==	Equal to	7 == 7	3 == 9
<=	Less than or equal to	5 <= 5	8 <= 6
>=	Greater than or equal to	7 >= 3	1 >= 2
!=	Not equal to	5 != 6	3 != 3

Table 2-3 Relational operators



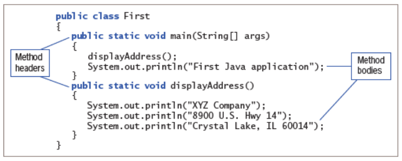
Escape Sequence	Description
\b	Backspace; moves the cursor one space to the left
\t	Tab; moves the cursor to the next tab stop
\n	Newline or linefeed; moves the cursor to the beginning of the next line
\r	Carriage return; moves the cursor to the beginning of the current line
\"	Double quotation mark; displays a double quotation mark
\'	Single quotation mark; displays a single quotation mark
\\	Backslash; displays a backslash character

Table 2-6 Common escape sequences

Ch. 3 – Using Methods // **Class header**: (3 parts) access Class identifier: public Class ExampleClass // **Method Signature**: method name + the parameters it has

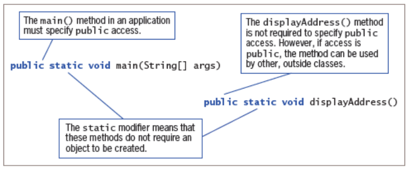
- The main method will always execute first!
- **Unreachable statements**: (dead code) Code that is written after a return statement, creates a compile error

- **Method header**: (4 parts) : access staticModifier(optional) return signature: public static void method1(int a)



```
public class First
{
   public static void main(String[] args)
   {
      displayAddress();
      System.out.println("First Java application");
   }
   public static void displayAddress()
   {
      System.out.println("XYZ Company");
      System.out.println("8900 U.S. Hwy 14");
      System.out.println("Crystal Lake, IL 60014");
   }
}
```

Figure 3-6 The headers and bodies of the methods in the First class



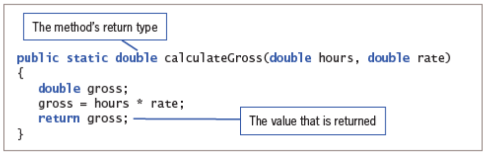
```
public static void main(String[] args)

                          public static void displayAddress()
```

The main() method in an application must specify public access.

The displayAddress() method is not required to specify public access. However, if access is public, the method can be used by other, outside classes.

The static modifier means that these methods do not require an object to be created.

Figure 3-7 Access specifiers for two methods



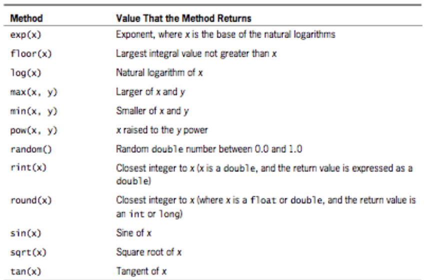
```
public static double calculateGross(double hours, double rate)
{
   double gross;
   gross = hours * rate;
   return gross;
}
```

Figure 3-17 A version of the calculateGross() method that returns a double

Ch. 4 – More Object Concepts // Don't assume that a constant is still a constant when passed to a method's parameter, use *final*. // **Gregorian Calendar returns int only!! Starts at 0 for January**

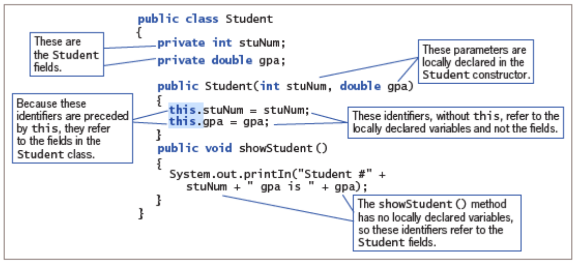
- **Reference**: variable name is a reference to a memory location. When in doubt the answer is: MEMORY LOCATION
- **Overriding**: child class method has the same name as a parent class method, you will override and use the child one.
- **Overloading**: Using the same method name with different parameters
- **Ambiguous**: Java doesn't know which method to use, if there is ambigiouty you will have either a compile error or a runtime error

- **Math Class**: To use, declare the full class for each method Java.lang.Math.method() or Java.lang.Math.CONSTANT
- **Methods that have identical names and parameter, but with different return types are illegal!**
- If a class's only constructor requires an argument, every object should have one.
- If you call *this* from a constructor, it must be the first statement within the constructor.
- The java.lang package is the only automatically imported, name package



Method	Value That the Method Returns
exp(x)	Exponent, where x is the base of the natural logarithms
floor(x)	Largest integral value not greater than x
log(x)	Natural logarithm of x
max(x, y)	Larger of x and y
min(x, y)	Smaller of x and y
pow(x, y)	x raised to the y power
random()	Random double number between 0.0 and 1.0
rint(x)	Closest integer to x (x is a double, and the return value is expressed as a double)
round(x)	Closest integer to x (where x is a float or double, and the return value is an int or long)
sin(x)	Sine of x
sqrt(x)	Square root of x
tan(x)	Tangent of x

Table 4-1 Common Math class methods



```
public class Student
{
   private int stuNum;
   private double gpa;

   public Student(int stuNum, double gpa)
   {
     this.stuNum = stuNum;
     this.gpa = gpa;
   }
   public void showStudent ()
   {
     System.out.printIn("Student #" +
        stuNum + " gpa is " + gpa);
   }
}
```

These are the Student fields.

These parameters are locally declared in the Student constructor.

Because these identifiers are preceded by this, they refer to the fields in the Student class.

These identifiers, without this, refer to the locally declared variables and not the fields.

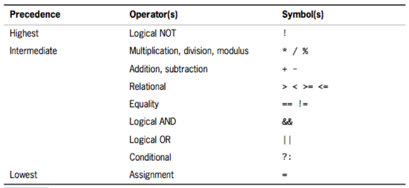
The showStudent () method has no locally declared variables, so these identifiers refer to the Student fields.

Figure 4-28 The Student class using the explicit this reference within the constructor

Ch. 5 – Making Decisions // You can code an *if* without an *else*, but it is illegal to code an *else* without an *if* that precedes it. Don't place a ';' in an if statement! // Make sure to use two == instead of one!

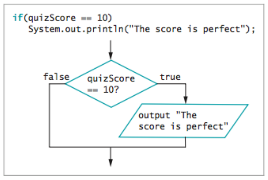
- When you use the && operator, you must include a complete Boolean expression on each side!!
- Don't forget the curly braces. If else statements are "first in, first out" basis.
- **Conditional operator**: shortcut if statement: boolean ? trueValue : falseValue   example: 2<3 ? a : b; will return a
- **Switch statement**: Useful with Int, char, String data types

- ***Switch:*** starts the structure and is followed by a test expression enclosed in parentheses
- ***Case:*** keyword case followed by test then a colon: case 1 or case 'a': or case "string":
- ***Break***: after each case, terminates the switch statement: Break;
- ***Default:*** if none of the case tests are true, will run the defualt case: Default:



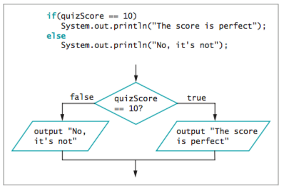
Precedence	Operator(s)	Symbol(s)
Highest	Logical NOT	!
Intermediate	Multiplication, division, modulus	* / %
	Addition, subtraction	+ -
	Relational	> < >= <=
	Equality	== !=
	Logical AND	&&
	Logical OR	||
	Conditional	?:
Lowest	Assignment	=

Table 5-1 Operator precedence for operators used so far



```
if(quizScore == 10)
   System.out.println("The score is perfect");
```

Figure 5-3 A Java if statement

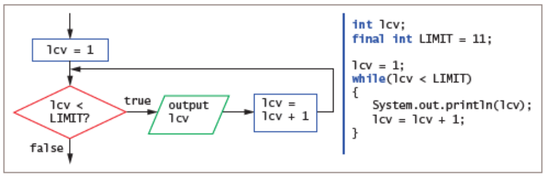


```
if(quizScore == 10)
   System.out.println("The score is perfect");
else
   System.out.println("No, it's not");
```

Figure 5-5 An if…else structure

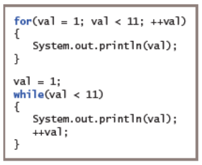
Ch. 6 – Looping // best structure to validate input data // ++prefix or postfix++ on constants <u>cannot</u> be used (Is ++84 legal?)

- ***while*** loop, boolean expression is first statement in the loop (first, second, third, last) while(boolean){code}
- ***for*** loop, used as a concise format in which to execute loops: for(initialize; test; LCV)
- ***do…while*** loop, boolean expression is <u>last</u> statement (first, second, third, last) do{code}while(boolean)
- <u>To prevent infinite looping</u>: loop control variable is initialized to a starting value, tested LCV in while statement, LCV altered within the body of the loop (eventually false, so stops)
- When you use the ++prefix, change the variable BEFORE the line is run
- when you use postfix++, change the variable AFTER the line is run



```
int lcv;
final int LIMIT = 11;

lcv = 1;
while(lcv < LIMIT)
{
   System.out.println(lcv);
   lcv = lcv + 1;
}
```

Figure 6-2 A while loop that displays the integers 1 through 10



```
for(val = 1; val < 11; ++val)
{
   System.out.println(val);
}

val = 1;
while(val < 11)
{
   System.out.println(val);
   ++val;
}
```

Figure 6-18 A for loop and a while loop that display the integers 1 through 10

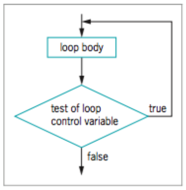
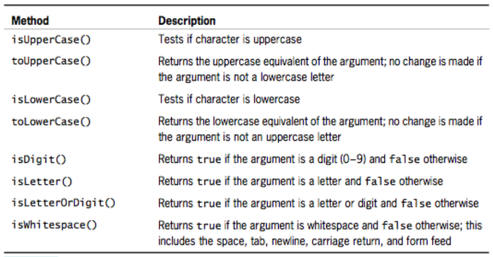


Figure 6-20 General structure of a do…while loop

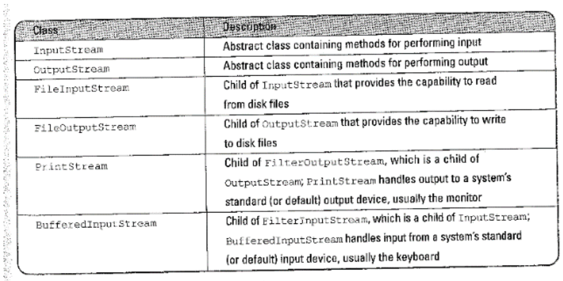
Ch. 7 – Characters, Strings, and the StringBuilder // a string is a **reference**: a variable that holds a memory address // **indexOf() method**: returns position // **charAt() method**: returns char

- **equals() method**: evaluates the contents of two String objects to determine if they are equivalent
- <u>Every character counts and has to be exact in order for this method to be true.</u>
- **equalsIgnoreCase() method**
- **compareTo() method**: compares two Strings by providing an integer value (0 if same, negative is "less", positive if "more"
- **length() method**: returns the length of the string
- **endsWith()** & **startsWith() method**: case sensitive, returns T or F
- **replace() method:** replaces portion of string with another string ONCE
- **toString() method**: converts primitive data types to Strings
- **substring():** takes portion of a string, arguments are start position and end position, does not include end position
- **regionMatches():** checks if a portion of one string matches another



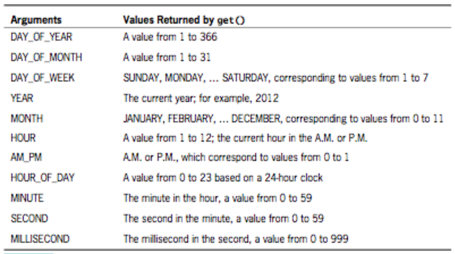
Method	Description
isUpperCase()	Tests if character is uppercase
toUpperCase()	Returns the uppercase equivalent of the argument; no change is made if the argument is not a lowercase letter
isLowerCase()	Tests if character is lowercase
toLowerCase()	Returns the lowercase equivalent of the argument; no change is made if the argument is not an uppercase letter
isDigit()	Returns true if the argument is a digit (0–9) and false otherwise
isLetter()	Returns true if the argument is a letter and false otherwise
isLetterOrDigit()	Returns true if the argument is a letter or digit and false otherwise
isWhitespace()	Returns true if the argument is whitespace and false otherwise; this includes the space, tab, newline, carriage return, and form feed

Table 7-1 Commonly used methods of the Character class



Class	Description
InputStream	Abstract class containing methods for performing input
OutputStream	Abstract class containing methods for performing output
FileInputStream	Child of InputStream that provides the capability to read from disk files
FileOutputStream	Child of OutputStream that provides the capability to write to disk files
PrintStream	Child of FilterOutputStream, which is a child of OutputStream; PrintStream handles output to a system's standard (or default) output device, usually the monitor
BufferedInputStream	Child of FilterInputStream, which is a child of InputStream; BufferedInputStream handles input from a system's standard (or default) input device, usually the keyboard

Table 12-2 Description of selected classes used for input and output



Arguments	Values Returned by get()
DAY_OF_YEAR	A value from 1 to 366
DAY_OF_MONTH	A value from 1 to 31
DAY_OF_WEEK	SUNDAY, MONDAY, … SATURDAY, corresponding to values from 1 to 7
YEAR	The current year; for example, 2012
MONTH	JANUARY, FEBRUARY, … DECEMBER, corresponding to values from 0 to 11
HOUR	A value from 1 to 12; the current hour in the A.M. or P.M.
AM_PM	A.M. or P.M., which correspond to values from 0 to 1
HOUR_OF_DAY	A value from 0 to 23 based on a 24-hour clock
MINUTE	The minute in the hour, a value from 0 to 59
SECOND	The second in the minute, a value from 0 to 59
MILLISECOND	The millisecond in the second, a value from 0 to 999

Table 4-2 Some possible arguments to and returns from the GregorianCalendar get() method

Ch. 8 – **Arrays**: named list of data items that all have the same type // **element**: one variable or object in an array // **subscript**: integer contained within square brackets that is an element

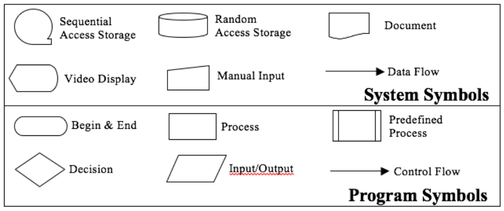
- **Length field**: number of elements in an array
- To **populate an array**: either initialize every element or none of them
- **parallel array**: same # of elements as another that corresponds
- The lowest array subscript is 0 and the highest is one less than the length of the array.
- Don't forget the ; after the closing curly braces in an array initialization list.
- <u>Cannot compare</u> ==

Ch. 10 – Introduction to Inheritance// Benefits: saves time bc fields and methods already exist, reduce errors bc methods are already used and tested, reduce the amt. Of new learning required

- **Composition**: members would <u>not</u> continue to exist // **Aggregation**: members would continue to exist // Inheritance is a 1-way proposition; a child inherits from a parent only
- **instanceof operator**: determines whether an object is a member or descendent of a class (true/false) // <u>Superclass constructor must execute first, and then the subclass executes</u>
- **Override the method**: child class that has same name and parameter as method in its parent class // <u>the super() statement must be the first statement in any subclass constructor</u>
- Members in a subclass can use all of the data fields and methods from its parents, except: private members of the parent class are not accessible within a child class's methods.
- <u>Methods you cannot override in a subclass</u>: static methods, final methods, methods within final classes (you can declare a class to be final, but the final cannot be a parent)

Ch. 12 – Files // **random access memory**: (RAM) temporary storage// **volatile**: value stored is lost when program ends

- **Close the file**: failure to close a <u>writing data</u> will cause inaccessible data, but <u>reading data</u> is fine
- **Writing Formatted File Data**: use DataInputStream and DataOutputStream
- **Chaining stream objects**: using DataOutputStream connected to FileOutputStream (out = new DataOutputStream(new FileOutputStream ("someFile")); - writes to a file
- <u>Using a variable Filename</u>: pass the name to the method that opens the file, use the args parameter//**Coincidental Cohesion**: no meaningful relationship among module statements (<u>lowest</u>)


Sequential Access Storage
Random Access Storage
Document
Video Display
Manual Input
Data Flow
System Symbols
Begin & End
Process
Predefined Process
Decision
Input/Output
Control Flow
Program Symbols