
ORACLE
ACADEMY

Java Fundamentals

4-3
Data Types and Operators

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Use primitive data types in Java code
- Specify literals for the primitive types and for Strings
- Demonstrate how to initialize variables
- Describe the scope rules of a method
- Recognize when an expression requires a type conversion

Overview

This lesson covers the following topics:

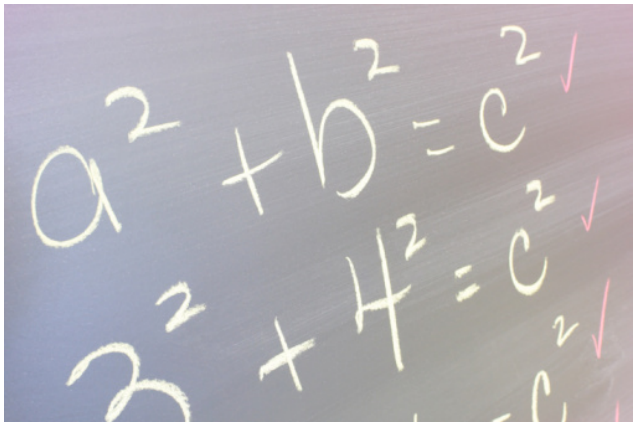
- Apply casting in Java code
- Use arithmetic operators
- Use the assignment operator
- Use a method from the Math class
- Access a Math class method from the Java API

Java Programming Types

- In Java, data types:
  - Are used to define the kind of data that can be stored inside a variable.
  - Ensure that only correct data is stored.
  - Are either declared or inferred.
  - Can be created by the programmer.

Variables Must Have Data Types

- All variables must have a data type for security reasons.
- The program will not compile if the user attempts to store data that is not the correct type.
- Programs must adhere to type constraints to execute.
  - Incorrect types in expressions or data are flagged as errors at compile time.



Primitive Data Types

- Java has eight primitive data types that are used to store data during a program's operation.
- Primitive data types are a special group of data types that do not use the keyword new when initialized.
- Java creates them as automatic variables that are not references, which are stored in memory with the name of the variable.
- The most common primitive types used in this course are int (integers) and double (decimals).

Primitive Data Types

Data Type	Size	Example Data	Data Description
boolean	1 bit	true, false	Stores true and false flags.
byte	1 byte (8 bits)	12, 128	Stores integers from -128 to 127.
char	2 bytes	'A', '5', '#'	Stores a 16-bit Unicode character from 0 to 65,535
short	2 bytes	6, -14, 2345	Stores integers from -32,768 to 32,767.

Primitive Data Types

Data Type	Size	Example Data	Data Description
int	4 bytes	6, -14, 2345	Stores integers from: -2,147,483,648 to 2,147,483,647
long	8 bytes	3459111, 2	Stores integers from: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	3.145, .077	Stores a positive or negative decimal number from: 1.4023×10^{-45} to $3.4028 \times 10^{+38}$
double	8 bytes	.0000456, 3.7	Stores a positive or negative decimal number from: 4.9406×10^{-324} to 1.7977×10^{308}

Declaring Variables and Using Literals

- The keyword new is not used when initializing a variable of a primitive type. Instead, a literal value should be assigned to each variable upon initialization.
- A literal can be any number, text, or other information that represents a value.
- Examples of declaring a variable and assigning it a literal value:

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

Declaring Variables and Using Literals Example

- The values of d1 and d2 are the same.
- The initialization of d2 shows how scientific notation can be used to set the value.
- Total and ss_num are assigned the same value.
- The initialization of ss_num shows that underscores can be used to separate numbers for readability.

```
long total=999999999;
long ss_num = 999_99_9999;
double d1 = 123.4;
double d2 = 1.234e2;
float f1  = 123.4f;
```

Numeric Literal Examples

- 0x and 0b are used to denote a literal hexadecimal value or a literal binary value.
- Literals will improve processing performance.

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_fff_fff_fffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Java SE 7 Binary Literals

- In Java SE7 and later, binary literals can be expressed using the binary system by adding the prefixes 0b or 0B to the number.
- Binary literals are Java int values. Java byte and short values require a cast to prevent a precision loss error from the compiler.

A cast signals a type conversion. If you cast one type as another type, you are explicitly converting the item from it's original type to the type that you specify. A cast does not round a decimal number, but instead, will truncate it. An example of casting a double as an int is:

```
double x = 5.745
int y = (int)x;  //y is now equal to 5
```

Pupose of Java SE 7 Binary Literals

- Binary Literals are used for:
  - Calculations
  - Comparisons
  - Low-level programming, such as:
    - Writing device drivers
    - Low-level graphics
    - Communications protocol packet assembly
    - Decoding

Why Use Binary Literals?

- Using Binary Literals to represent values for comparisons and calculations is substantially faster than using values of the actual data type.
- Modern high-performance processors usually perform calculations on integers as fast as using binary literals, so why use literals?
- It is still optimal to use literals for overall power and performance because they use less resources.

Casting Example

```
// An 8-bit 'byte' value:
byte aByte = (byte)0b00100001;

// A 16-bit 'short' value:
short aShort = (short)0b1010_0001_0100_0101;

// Some 32-bit 'int' values:
int anInt1=0b1010_0001_0100_0101_1010_0001_0100_0101;
int anInt2=0b101;
int anInt3=0B101;                    //The B can be upper or lower case
```

This is an example of casting. In this example, the binary value is cast to a byte type.

Rules for Variable Names

- You must follow the following rules when choosing the name for a variable:
  - Do not use a Java keyword or reserved word.
  - Do not use a space in the variable name.
  - Use a combination of letters or a combination of letters and numbers.
  - Cannot start with a number.
  - The only symbols allowed are the underscore (_) and the dollar sign ($).

Conventions for Variable Names

- While conventions are not rules, most Java programmers follow these conventions:
  - Use full words instead of cryptic abbreviations.
  - Do not use single letter variables. If all of the variables are single letter, the code may look very confusing.
    - An exception to this convention is for loop control variables, which are often letters i, j, or k
  - If a variable name consists of one word, spell that word in all lowercase letters.
  - If a variable name consists of more than one word, use lowerCamelCase.

Additional Naming Conventions for Variable Names

- Additional naming conventions:
  - If a variable will be a constant value, use all capital letters and separate words with the underscore.
  - Use names that express the purpose of the variable.
  - In the example below, PI is a good choice for naming this number because it allows you to recall what the variable is.

```
double PI = 3.14159;
```

Variable Scope

- Scope is used to describe the block of code where a variable exists in a program.
- It is possible for multiple variables with the same name to exist in a Java program.
  - In most cases, the innermost variable has precedence.
- A variable exists only inside the code block in which it is declared.
- Once the final brace of the block } is reached:
  - The variable goes out of scope.
  - It is no longer recognized as a declared variable.

Variable Scope Example 1

- In the following example, name will not print out because it stops existing once the brace marked Point B is reached.

```
public void someMethod()
   {
     if(gameOver && score>highScore)
     {
       String name;                              //Point A
       System.out.println("Please enter your name:");
         name=reader.next();
     }                                           //Point B

     System.out.println("Thank you " + name + ", ");
     System.out.println("your high score has been saved.");
   }
```

Variable Scope Example 2

- In this example, the variable name has been moved outside of the if statement block to allow name to be used throughout the method.

```
public void someMethod()
  {
    String name;                        //Point A
    if(gameOver && score>highScore)
    {
      System.out.println("Please enter your name:");
      name=reader.next();
    }                                   //Point B
    System.out.println("Thank you " + name + ", ");
    System.out.println("your high score has been saved.");
  }
```

Variable Scope Example 3

- Java will allow a class variable and a method variable with the same name to exist in a program. Can you predict what this program will print?

```
public class Counting{
  public static int counter=5;
  public static void main(String[] args)
  {
    System.out.println("At the start of this program, counter is "+ counter);
    count();
    System.out.println("At the end of this program, counter is "+ counter);
  }//end of main
  public static void count()
  {
    int counter=10;
    System.out.println("At the end of this method, counter is "+ counter);
  }//end of count()
}//end of class
```

Variable Scope Example 3 Solution

- Solution: 5, 10, 5
- The program starts main() and prints the global class variable counter.
- The method count() is called and a new local variable counter is created for that method call.
- It is given a value of 10 and then prints.
- When the brace at the end of count() is reached, the local variable goes out of scope (ceases to exist).
- The program returns to the main() method and prints the global variable counter which has not changed it's value.

Arithmetic Operators

- Java has several arithmetic operators to perform math operations.

Symbol	Operator Description
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator (finds the quotient)
%	Modular operator (finds the remainder)
+ +	Increment operator (adds one). Is a unary operator.
- -	Decrement operator (subtracts one). Is a unary operator.

Arithmetic Operators Precedence

- All math expressions are evaluated following the order of precedence:
  - Expressions in parenthesis are handled first.
  - All multiplication, division, and modular operations are handled next, working from left to right.
  - Finally, all addition and subtraction operations are handled, working from left to right.

Increments and Decrements

- Increments and decrements are handled first for pre-increment notation and last for post-increment notation.
- Increment in Java means to add one to the variable value.
- Decrement in Java means to subtract one from the variable.
- Pre-increment notation:

```
++x;
```

- Post-increment notation:

```
x++;
```

Increment and Decrement Precedence Example

- Pre-increment notation:

```
int x = 3;
++x;          //x is equal to 4
z = ++x;      //x is equal to 5, THEN z is equal to 5
```

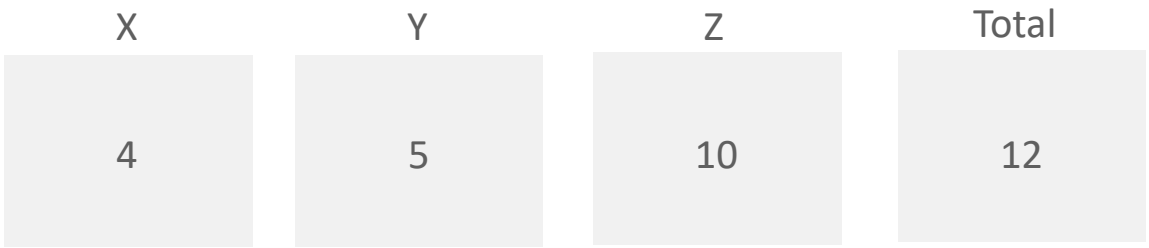
- Post-increment notation:

```
int x = 3;
x++;          //x is equal to 4
z = x++;      //z is equal to 4, THEN x is equal to 5
```

Assignment Operator

- Java uses the = (equal sign) as the assignment operator. The evaluation of the expression on the right is assigned to the memory location on the left.

```
int x = 4;
int y = 5;
int z = 10;
int total = 12;
```



Assignment Operator Example 1

- When this line of code is executed, the value for total changes. The boxes show what is in each memory location for x, y, z, and total. Now the memory location total is assigned the value of 4 + 5 * 10, which is 54.

```
int x = 4, y = 5, z = 10;
int total = x + y * z;
```

X	Y	Z	Total
4	5	10	54

Assignment Operator Example 2

- Think of the assignment operator like an arrow pointing to the left. Everything on the right will go into the memory location on the left. How will memory change when this code is executed?

```
int x = 4, y = 5, z = 10;
int total = x + y * (z - x);
```

X	Y	Z	Total
4	5	10	??

Assignment Operator Example 2 Solution

- The answer is that total will be assigned the value of the expression. This means that the value of the expression will be stored in the memory address associated with the variable total.

```
int x = 4, y = 5, z = 10;
int total = x + y * (z - x);
```

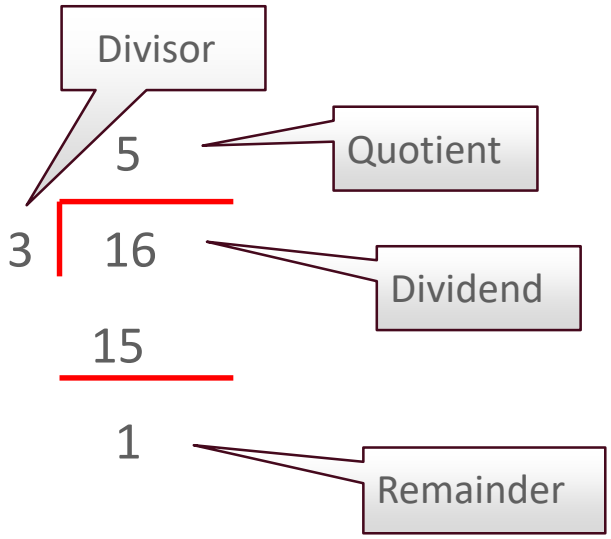
X	Y	Z	Total
4	5	10	34

Truncation and Integer Division

- Division of two integers will ALWAYS produce an integer.
- For example, the formula for the volume of a cone from Geometry is:
  - V = ⅓ r2 h
  - If 1/3 is used in a Java expression, it is evaluated as 0 because of integer division.
  - Integer division results are the quotient without decimals.

Truncation is the concept of removing the fractional or decimal part of a number. For example: Truncating 7.8 would produce a result of 7, and truncating -3.2 would produce a result of -3.

Truncation and Integer Division



Integer Division Examples:

- 16 / 3 = 5 (Truncation)
- 10 / 3 = 3 (Truncation)
- 16 % 3 = 1
- 12 % 3 = 0
- 11 % 3 = 2

- To calculate a quotient without truncation (decimal results), convert either the dividend or the divisor to a decimal.
- For example: 11 / 5.0 = 2.2 and likewise, 11.0 / 5 = 2.2

Truncation and Integer Division Example 1

- What prints when the code below is executed?

```
int x = 4, y = 5, z = 10;
int total = z / (x * y);

System.out.println("The total is " + total + ".");
```

- The answer is 0. Why doesn't 0.5 display?
- Since total is an integer, the system will store an integer value as the result of the calculation.
- The decimal portion of the answer is truncated rather than rounded to produce the final integer answer of 0.

Truncation and Integer Division Example 2

- A programmer might write the following code to create a program to calculate volume.
- The Java program will incorrectly calculate the answer as 0.
- Working from left to right, the program divides 1 by 3.
- Java considers 1 and 3 to be literal integers and does integer division where .33... is truncated to 0.
- How would you correct this?

```
double height = 4, radius = 10, volume;
volume = 1 / 3 * 3.14 * radius * radius * height;
System.out.println("The volume is" + volume + ".");
```

Understanding Types and Conversions

- There are a few ways to force a formula to not truncate a value:
  - Move the fraction to the end so that Java will always use a double and an integer and will implicitly convert the answer to a double, not truncate.

```
double volume = 3.14 * radius * radius * height * 1 / 3;
```

  - Make one of the literal integers into a literal double so that Java will always use a double and an integer and will implicitly convert the answer to a double, not truncate.

```
double volume = 1 / 3.0 * 3.14 * radius * radius * height;
```

Implicit Type Conversions

- In the previous example, Java did implicit type conversions.
- This happens whenever a smaller data type (like int) is placed into a larger type (like double).
- Java realizes the types are different and converts to the larger size automatically for you.
- However, Java will not convert from a larger (like double) to a smaller (like int) size automatically.

Using Type Casting

- Using the random method from the Math library, we can generate a random number from 1 to 10.
- The random method generates a double between 0 and (not including) 1.
- Values such as 0, 0.4567 or 0.901306 might be generated.

```
int number;
number = Math.random() * 10;
System.out.println("The random number is " + number + ".");
```

Using Type Casting

- Multiplying these values by 10 and then truncating the extra would yield values 0, 4 or 9.
- However, Java will not let this program compile in its current state.
- Data is lost by going from a larger value (double) to a smaller value (int).
- Thus, type casting is required for this type conversion.

```
int number;
number = Math.random() * 10;
System.out.println("The random number is " + number + ".");
```

Type Casting Operator

- To cast a double value to an int, use (int) in front of the value.
- To get the double result from our formula to go into the integer container, use the type casting operator (int) in front of the value.
- Casting to the int data type will truncate the value.
- Thus, casting the double literal 4.567 to an int will result in 4, and 9.01306 will result in 9.

```
int number;
number = (int)(Math.random() * 10);
System.out.println("The random number is " + number + ".");
```

Converting Data Types

- You can convert a data type (primitive or reference) to another data type by simply placing the name of the data type in parenthesis in front of the value or variable, as shown in the example below.

```
int number;
Object o;
char firstInitial = 'A';
number = (int)firstInitial;
o = (Object)firstInitial;
```

Converting String Data Types

- Note that casting will not work in all situations.
- For example, casting a char to a String results in a compiler error.
- In situations such as this, you would need to resort to making the type conversion in another way.
- There are methods in the java.lang library to convert characters to strings.

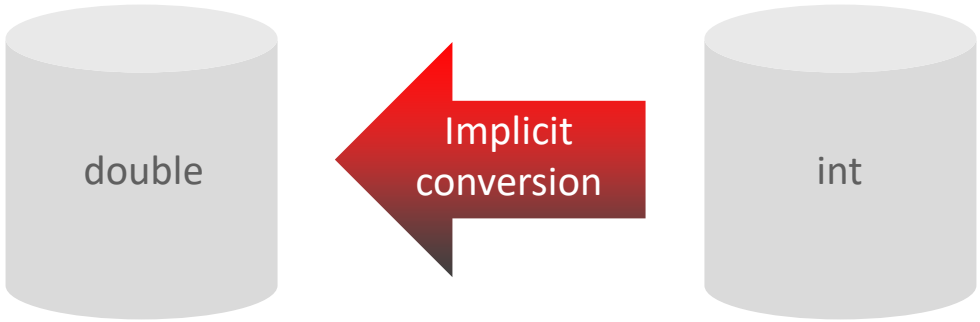
Using Type Conversions

- Using type conversions is another option to fixing the truncation issue with the volume formula shown previously.
- Use type casting to make one of the literal integers a double.

```
double volume = (double) 1 / 3 * 3.14 * radius * radius * height;
```

Understanding Types and Conversions

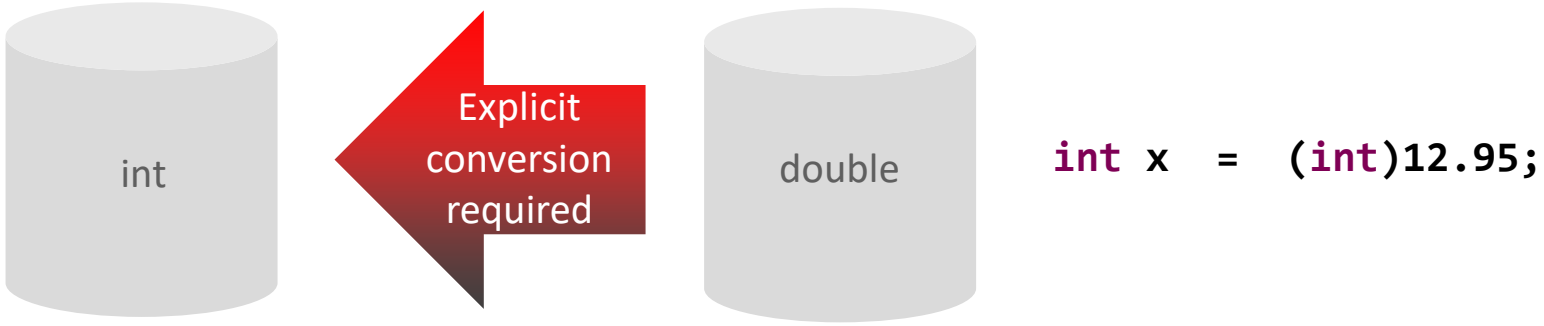
- When Java is converting from a smaller primitive type to a larger primitive type, the conversion is implicit.




```
double  d1  =  5;
```

Understanding Types and Conversions

- However, when Java converts from a larger primitive type to a smaller primitive type, the conversion must be explicit via type casting.
- Java will not implicitly cast a larger type to a smaller type because of loss of precision.




```
int x  =  (int)12.95;
```

Searching Through the Java API

- Examples and exercises in this course will require the use of the methods in the Java Math and String classes.
- You can find a description of all Java methods in the online Java API.
- Understanding how to navigate this vast library of standard methods and classes will aid you in writing Java programs and reusing code blocks that have already been created by others.

Why Use the Java API?

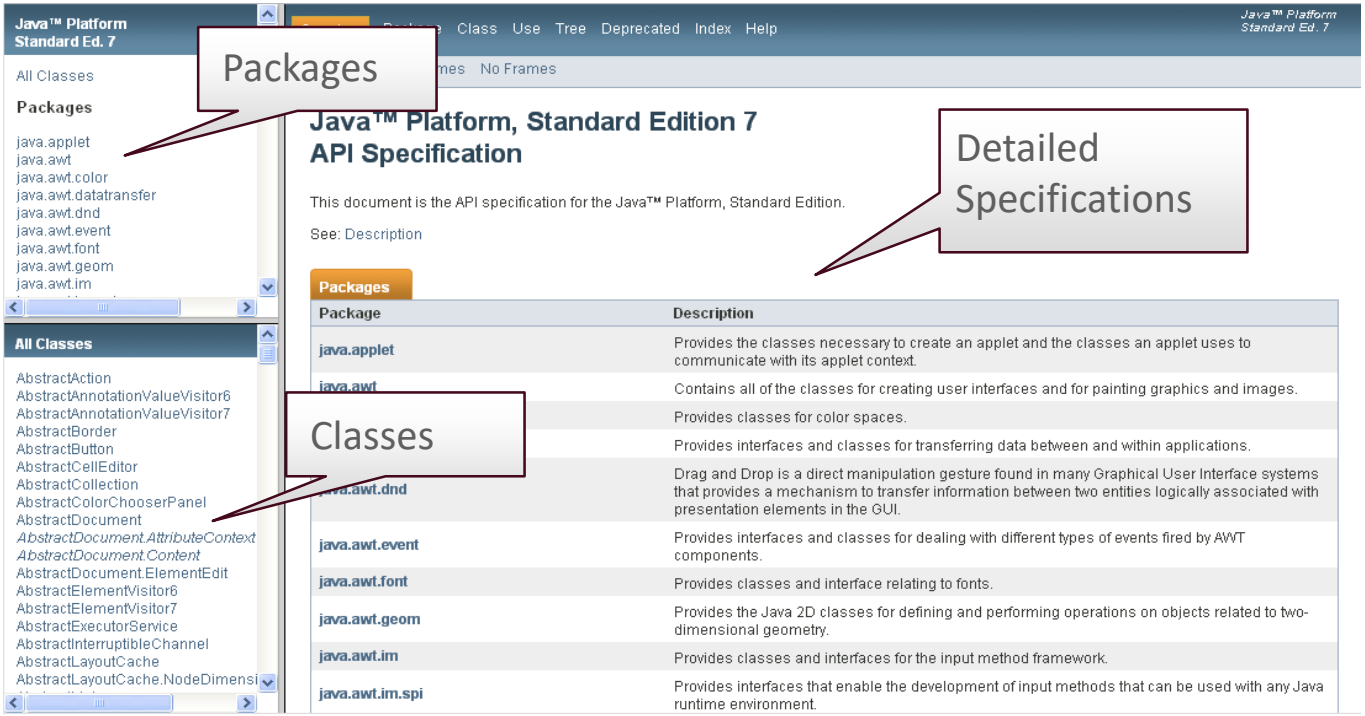
- One major benefit to having access to the Java API is a common concept for programmers called code reuse.
- Rather than coding excess items, you may use the API to find how to access existing code that does exactly what you want.
- This will reduce spending time on reproducing already existing code and make your programming much more efficient.

Review the Java API

- Go back to the Java API by using a search engine to search for it.
- There are many editions.
- Review the Standard Edition for Java 7: http://docs.oracle.com/javase/7/docs/api/
- Examine the Math class. See if you can find a value for PI and a method for computing the square root of a number.

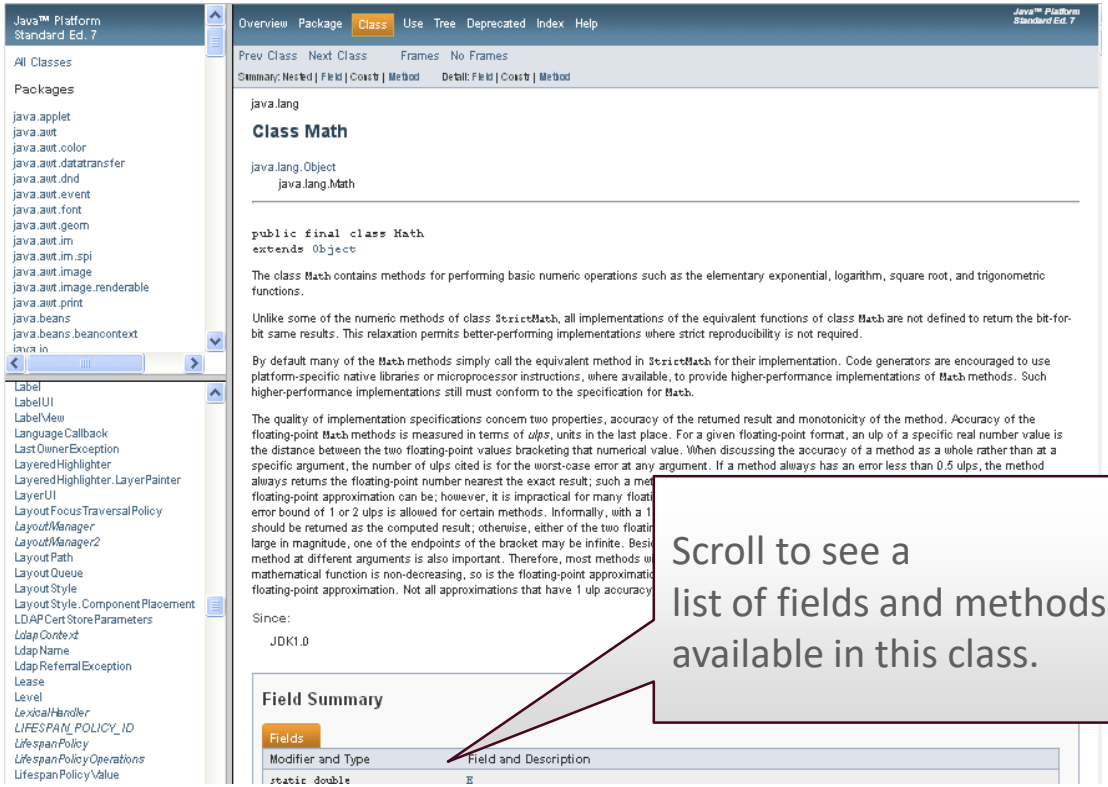
Math Class and the Java API

- The frames version of the API has windows for the packages, classes, and specifications.



Math Class

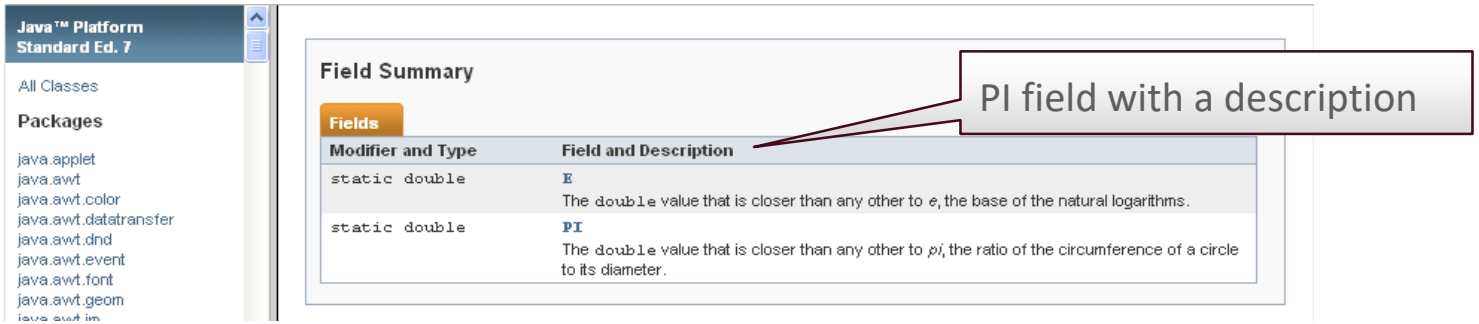
- Find the Math Class in the class frame window.



PI Field

- Scroll to find the PI field.
- To use PI in a program, specify the class name (Math) and PI separated by the dot operator.
- For example, this field would yield a more accurate volume calculation in our earlier example if it is used as follows:

```
double volume = (double) 1 / 3 * Math.PI * radius * radius * height;
```



Calculating Square Roots

- The method for calculating square roots:

`static double`	**sqrt**(double a) Returns the correctly rounded positive square root of a `double` value.

- To calculate the square root of 625, use the class name and the method separated again by the dot operator.

```
double answer = Math.sqrt(625);
```

- The sqrt method requires a double. Why does the literal integer 625 not give an error?

Calculating Square Roots Solution

- Question: The sqrt method requires a double.
- Why does the literal integer 625 not give an error?
- Solution: Implicit conversion.
- The int 625 is implicitly converted to a double and thus no error occurs.

Data Types and Operators Practice

- On paper, evaluate the following Java statements and record the result.

```
double x = 3.25;
double y = -4.5;
int m = 23;
int n = 9;
System.out.println(x + m * y - (y + n) * x);
System.out.println(m / n + m % n);
System.out.println(5 * x - n / 5);
System.out.println(Math.sqrt(Math.sqrt(n)));
System.out.println((int)x);
System.out.println(Math.round(y));
double x = 3.25;
double y = -4.5;
int m = 23;
int n = 9;
System.out.println((int)Math.round(x) + (int)Math.round(y));
System.out.println(m + n);
System.out.println(1-1-((1-(1-(1-n)))));
```

Terminology

Key terms used in this lesson included:

- Arithmetic operator
- Assignment operator
- boolean
- char
- Conventions
- Declaration
- double

Terminology

Key terms used in this lesson included:

- float
- Initialization
- int
- Literals
- long
- Order of Operation
- Primitive data types

Terminology

Key terms used in this lesson included:

- Scope
- Short
- Truncation
- Type casting
- Type conversion
- Variables

Summary

In this lesson, you should have learned how to:

- Use primitive data types in Java code
- Specify literals for the primitive types and for Strings
- Demonstrate how to initialize variables
- Describe the scope rules of a method
- Recognize when an expression requires a type conversion

Summary

In this lesson, you should have learned how to:

- Apply casting in Java code
- Use arithmetic operators
- Use the assignment operator
- Use a method from the Math class
- Access a Math class method from the Java API


ORACLE
ACADEMY