Manipulating and Formatting the Data in Your Program

7

Objectives

After completing this lesson, you should be able to:

- Describe the `String` class and use some of the methods of the `String` class
- Use the JDK API documentation to search for and learn how to use a class
- Describe the `StringBuilder` class
- Explain what a constant is and how to use it
- Explain the difference between promoting and casting of variables

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables

`String` Class

```
String hisName = "Fred Smith";
```

Standard syntax

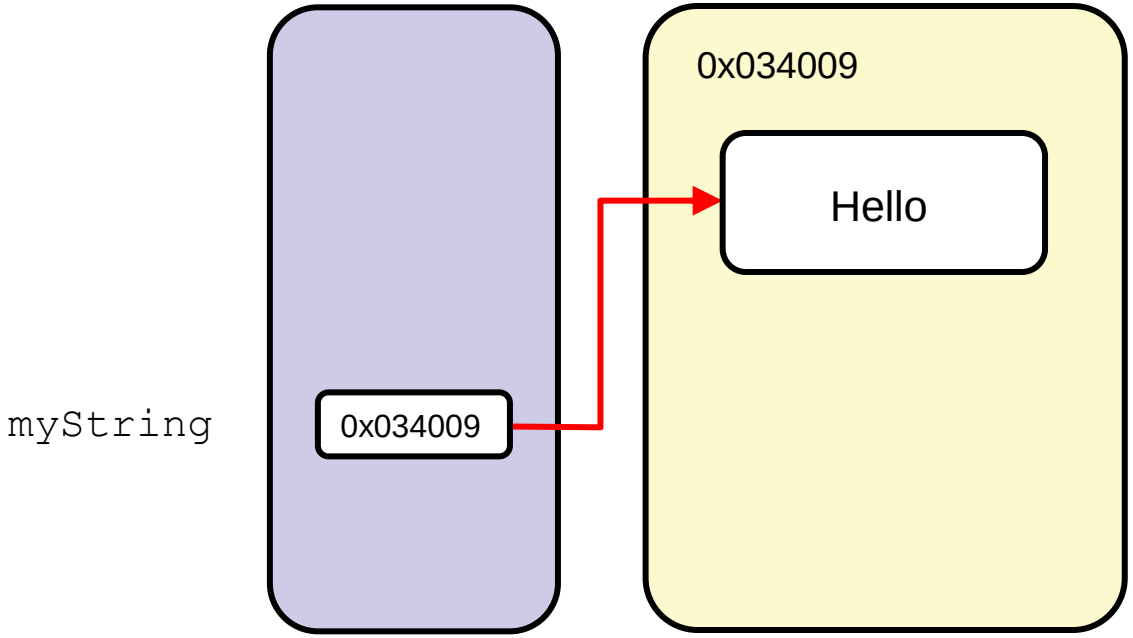
The `new` keyword can be used, but it is *not* best practice:

```
String herName = new String("Anne Smith");
```

- A `String` object is immutable; its value cannot be changed.
- A String object can be used with the string concatenation operator symbol (+) for concatenation.

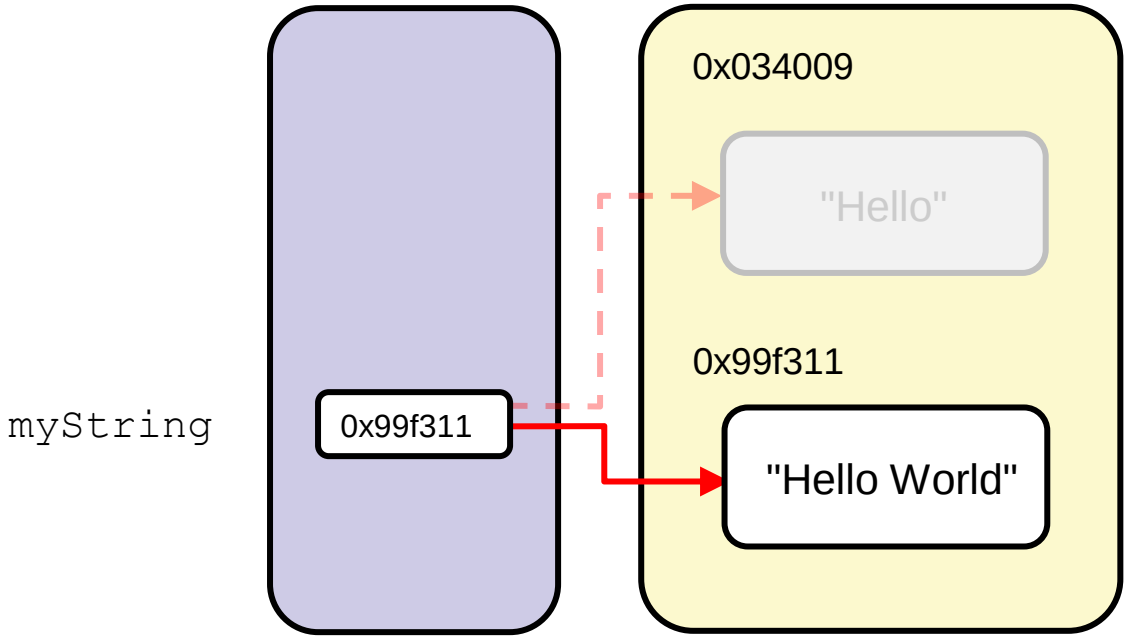
Concatenating Strings

```
String myString = "Hello";
```



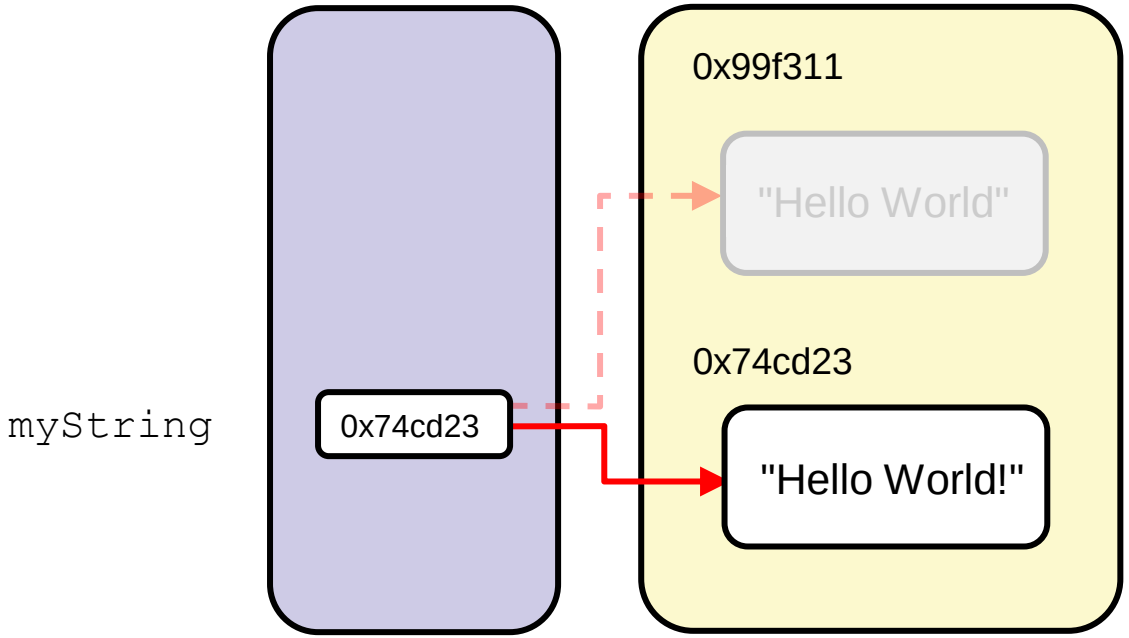
Concatenating Strings

```
String myString = "Hello";
myString = myString.concat(" World");
```



Concatenating Strings

```
String myString = "Hello";
myString = myString.concat(" World");
myString = myString + "!"
```



`String` Method Calls with Primitive Return Values

A method call can return a single value of any type.

- An example of a method of primitive type `int`:

```
String hello = "Hello World";
int stringLength = hello.length();
```

`String` Method Calls with Object Return Values

Method calls returning objects:

```
String greet = " HOW ".trim();
String lc = greet + "DY".toLowerCase();
```

Or

```
String lc = (greet + "DY").toLowerCase();
```

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables

Java API Documentation

Consists of a set of webpages;

- Lists all the classes in the API
  - Descriptions of what the class does
  - List of constructors, methods, and fields for the class
- Highly hyperlinked to show the interconnections between classes and to facilitate lookup
- Available on the Oracle website at: https://docs.oracle.com/en/java/javase/11/docs/api/index.html

JDK 11 API Documentation

Select one of the Module.

The packages for the selected module are listed here.

The classes for the selected package are listed here.

Java SE 11 & JDK 11

ALL CLASSES ALL PACKAGES

Modules

java.activation
java.base
java.compiler
java.corba

Java SE 9 & JDK 9

ALL CLASSES ALL PACKAGES
ALL MODULES

java.base Packages

java.io
java.lang
java.lang.annotation
java.lang.invoke
java.lang.module

StrictMath
String
StringBuffer
StringBuilder
System
System.LoggerFinder
Thread

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.base
Package java.lang

Class String

Details about the class selected

java.lang.Object
java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The String class represents character strings. All string literals in Java program

Strings are constant; their values cannot be changed after they are created. Stri

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Java Platform SE and JDK Version 11 API Specification

This document is divided into two sections:

- **Java SE:**
  - The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing.
  - These APIs are in modules whose names start with java.
- **JDK**
  - The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform.
  - These APIs are in modules whose names start with jdk.

Java Platform SE 11: Method Summary

```
public int charAt(String str)
```

The return type of the method

The name of the method

The type of the parameter that must be passed into the method

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
char	charAt(int index)	Returns the char value at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.
int	codePointAt(int index)	Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.

Java Platform SE 11: Method Detail

Click here to get the detailed description of the method.

int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Detailed description for the `indexOf()` method

indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring.

The returned index is the smallest value *k* for which:

```
this.startsWith(str, k)
```

If no such value of *k* exists, then `-1` is returned.

Parameters:

`str` - the substring to search for.

Returns:

the index of the first occurrence of the specified substring, or `-1` if there is no such occurrence.

Further details about parameters and return value are shown in the method list.

`indexOf` Method Example

```
String phoneNum = "404-543-2345";
int idx1 = phoneNum.indexOf('-');
System.out.println("index of first dash: "+ idx1);


int idx2 = phoneNum.indexOf('-', idx1+1);
System.out.println("second dash idx: "+idx2);
```

The 1-arg version

The 2-arg version

Exercise 7-1: Use `indexOf` and `substring` Methods

In this exercise, you get and display a customer's first name.

1. Open the project **`Exercise_07-1`** in NetBeans.
2. Use the `indexOf` method to get the index for the space character (" ") within `custName`. Assign it to `spaceIdx`.
3. Use the `substring` method and the `spaceIdx` to get the first name portion of `custName`.
   - Assign it to `firstName`.
   - Print `firstName`.

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables

`StringBuilder` Class

`StringBuilder` provides a mutable alternative to `String`. `StringBuilder`:

- Is instantiated using the `new` keyword
- Has many methods for manipulating its value
- Provides better performance because it is mutable
- Can be created with an initial capacity

`String` is still needed because:

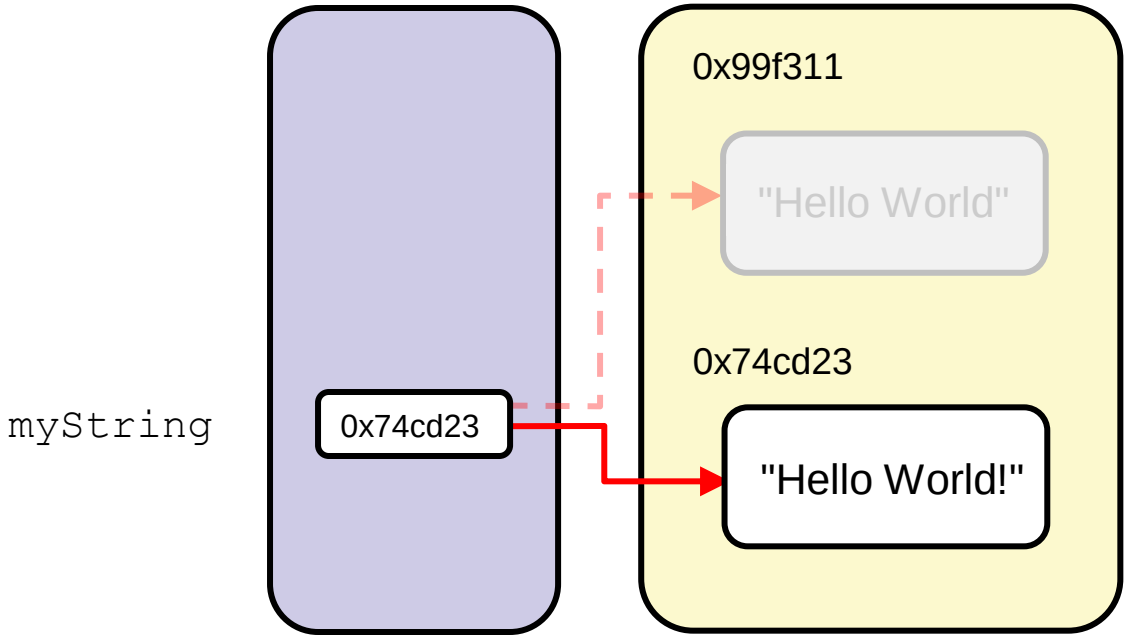
- It may be safer to use an immutable object
- A method in the API may require a string
- It has many more methods not available on `StringBuilder`

`StringBuilder` Advantages over String for Concatenation (or Appending)

String Concatenation

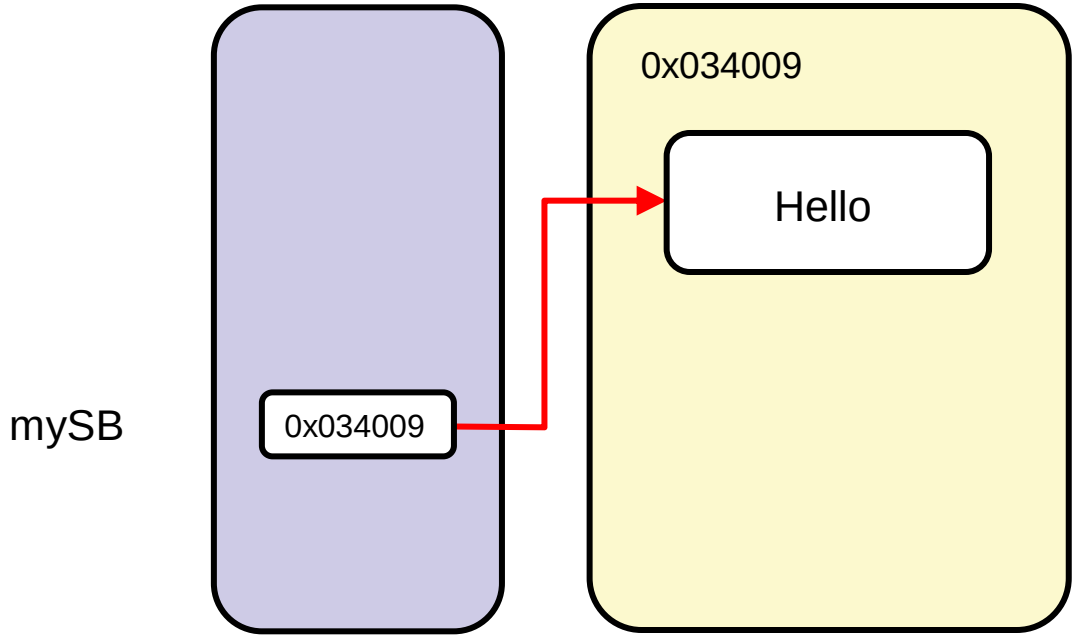
- Costly in terms of creating new objects

```
String myString = "Hello";
myString = myString + " World";
```



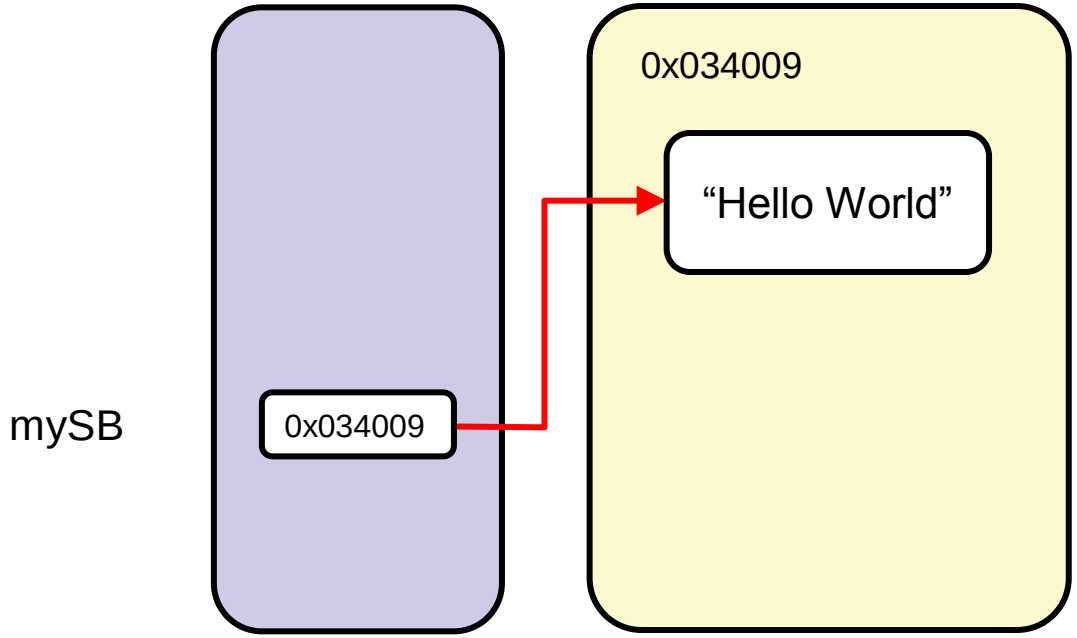
StringBuilder: Declare and Instantiate

```
StringBuilder mySB = new StringBuilder("Hello");
```



StringBuilder Append

```
StringBuilder mySB = new StringBuilder("Hello");
mySB.append(" World");
```



Quiz

Which of the following statements are true? (Choose all that apply.)

a. The dot (`.`) operator creates a new object instance.

b. The `String` class provides you with the ability to store a sequence of characters.

c. The Java API specification contains documentation for all of the classes in a Java technology product.

d. `String` objects cannot be modified.

Exercise 7-2: Instantiate the `StringBuilder` object

1. Open the project **`Exercise_07-2`** or continue editing the previous exercise.
2. Instantiate a `StringBuilder` object (`sb`), initializing it to `firstName`, using the `StringBuilder` constructor.
3. Use the `append` method of the `StringBuilder` to append the last name back onto the first name. You can just use a String literal for the last name. Print the `StringBuilder` object and test your code. It should show the full name.
4. (Optional) Can you append the last name without using a String literal?

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables

Primitive Data Types

- Integral types (`byte`, `short`, `int`, and `long`)
- Floating point types (`float` and `double`)
- Textual type (`char`)
- Logical type (`boolean`)

Some New Integral Primitive Types

Type	Length	Range
`byte`	8 bits	-2^7 to $2^7 - 1$ (–128 to 127, or 256 possible values)
`short`	16 bits	-2^{15} to $2^{15} - 1$ (–32,768 to 32,767, or 65,535 possible values)
`int`	32 bits	-2^{31} to $2^{31} - 1$ (–2,147,483,648 to 2,147,483,647, or 4,294,967,296 possible values)
`long`	64 bits	-2^{63} to $2^{63} - 1$ (–9,223,372,036854,775,808 to 9,223,372,036854,775,807, or 18,446,744,073,709,551,616 possible values)

Floating Point Primitive Types

Type	Float Length
`float`	32 bits
`double` (default type for floating point literals)	64 bits

Example:

```
public float pi = 3.141592F;
```

Textual Primitive Type

- The only primitive textual data type is `char`.
- It is used for a single character (16 bits).
- Example:
  - `public char colorCode = 'U';`

Single quotes must be used with char literal values.

Java Language Trivia: Unicode

- Unicode is a standard character encoding system.
  - It uses a 16-bit character set.
  - It can store all the necessary characters from most languages.
  - Programs can be written so they display the correct language for most countries.

Character	UTF-16	UTF-8	UCS-2
A	0041	41	0041
c	0063	63	0063
Ö	00F6	C3 B6	00F6
亜	4E9C	E4 BA 9C	4E9C
𝄞	D834 DD1E	F0 9D 84 9E	N/A

Constants

- Variable (can change):
  - `double salesTax = 6.25;`
- Constant (cannot change):
  - `final int NUMBER_OF_MONTHS = 12;`

The `final` keyword causes a variable to be read only.

Quiz

The variable declaration `public int myInteger=10;` adheres to the variable declaration and initialization syntax.

a. True

b. False

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables

Modulus Operator

Purpose	Operator	Example	Comments

Remainder	`%`
	modulus
		`num1` = 31;
		`num2` = 6;
		
		`mod = num1 % num2;`
		
		`mod is 1`
			Remainder finds the remainder of the first number divided by the second number.
			5 R 1
			6 ⟌ 31
			30
			--- 	
			1
			Remainder always gives an answer with the same sign as the first operand.

Combining Operators to Make Assignments

Purpose	**Operator**	**Examples** `int a = 6, b = 2;`	**Result**
Add to and assign	+=	a += b	a = 8
Subtract from and assign	-=	a -= b	a = 4
Multiply by and assign	*=	a *= b	a = 12
Divide by and assign	/=	a /= b	a = 3
Get remainder and assign	%=	a %= b	a = 0

More on Increment and Decrement Operators

Operator	Purpose	Example

`++`	Preincrement (`++variable`)	`int id = 6;`
		`int newId = ++id;`
		`id is 7, newId is 7`

	Postincrement (`variable++`)	`int id = 6;`
		`int newId = id++;`
		`id is 7, newId is 6`

`--`	Predecrement (`--variable`)	*(same principle applies)*
	Postdecrement (`variable--`)	

Increment and Decrement Operators (++ and −−)

Examples:

```
int count=15;
int a, b, c, d;
a = count++;
b = count;
c = ++count;
d = count;
System.out.println(a + ", " + b + ", " + c + ", " + d);
```

Output:

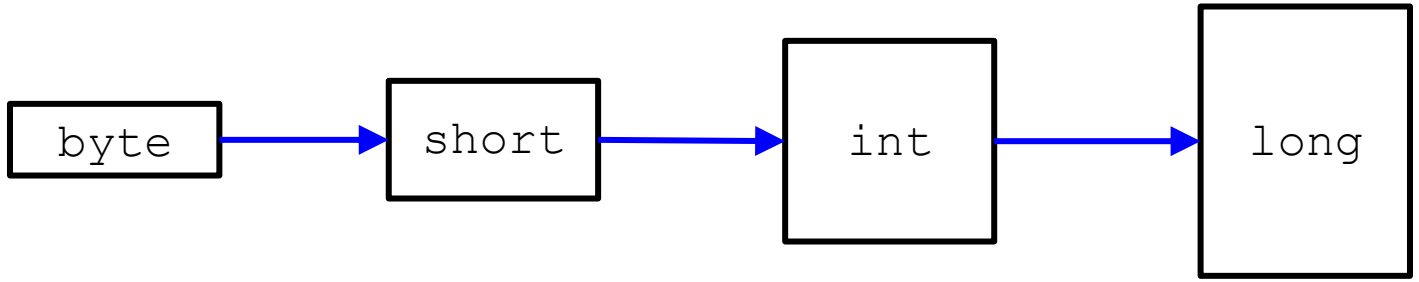
```
15, 16, 17, 17
```

Topics

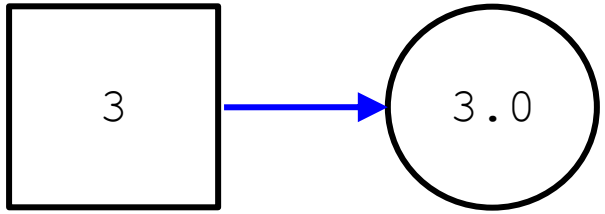
- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables

Promotion

- Automatic promotions:
  - If you assign a smaller type to a larger type



  - If you assign an integral type to a floating point type



- Examples of automatic promotions:
  - `long intToLong = 6;`
  - `double intToDouble = 3;`

Caution with Promotion

Equation:

55555 * 66666 = 3703629630

Example of potential issue:

```
int num1 = 55555;
int num2 = 66666;
long num3;
num3 = num1 * num2;          //num3 is -591337666
```

Example of potential solution:

```
int num1 = 55555;
long num2 = 66666;
long num3;
num3 = num1 * num2;          //num3 is 3703629630
```

Changed from int to long

Caution with Promotion

Equation:

7 / 2 = 3.5

Example of potential issue:

```
int num1 = 7;
int num2 = 2;
double num3;
num3 = num1 / num2;          //num3 is 3.0
```

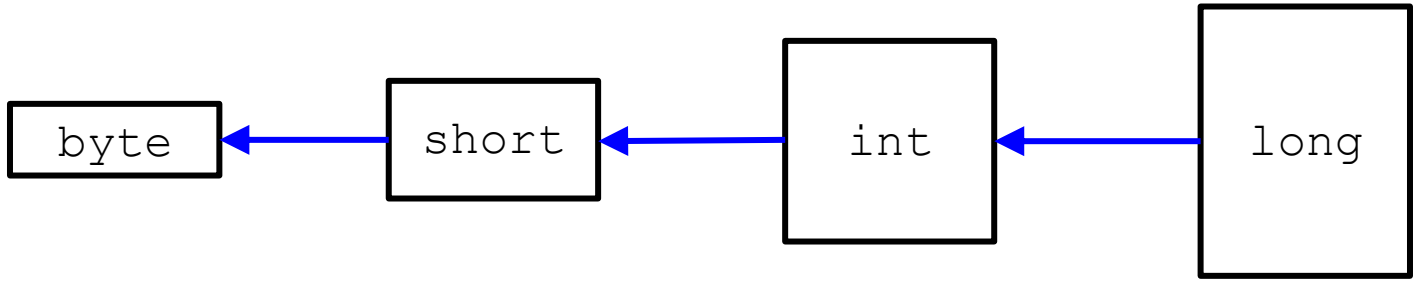
Example of potential solution:

```
int num1 = 7;
double num2 = 2;
double num3;
num3 = num1 / num2;          //num3 is 3.5
```

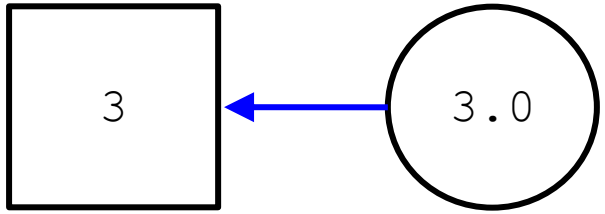
Changed from int to double

Type Casting

- When to cast:
  - If you assign a larger type to a smaller type



  - If you assign a floating point type to an integral type



- Examples of casting:
  - `int longToInt = (int)20L;`
  - `short doubleToShort = (short)3.0;`

Caution with Type Casting

Example of potential issue:

```
int myInt;
long myLong = 123987654321L;
myInt = (int) (myLong); // Number is "chopped"
                        // myInt is -566397263
```

Safer example of casting:

```
int myInt;
long myLong = 99L;
myInt = (int) (myLong); // No data loss, only zeroes.
                        // myInt is 99
```

Caution with Type Casting

- Be aware of the possibility of lost precision.

Example of potential issue:

```
int myInt;
double myPercent = 51.9;
myInt = (int) (myPercent);  // Number is "chopped"
                            // myInt is 51
```

Using Promotion and Casting

Example of potential issue:

```
int num1 = 53; // 32 bits of memory to hold the value
int num2 = 47; // 32 bits of memory to hold the value
byte num3;     // 8 bits of memory reserved
num3 = (num1 + num2); // causes compiler error
```

Solution using a larger type for num3:

```
int num1 = 53;
int num2 = 47;
int num3;
num3 = (num1 + num2);
```

Changed from byte to int

Solution using casting:

```
int num1 = 53; // 32 bits of memory to hold the value
int num2 = 47; // 32 bits of memory to hold the value
byte num3;     // 8 bits of memory reserved
num3 = (byte)(num1 + num2); // no data loss
```

Compiler Assumptions for Integral and Floating Point Data Types

- Most operations result in an `int` or `long`:
  - `byte`, `char`, and `short` values are automatically promoted to `int` prior to an operation.
  - If an expression contains a `long`, the entire expression is promoted to `long`.
- If an expression contains a floating point, the entire expression is promoted to a floating point.
- All literal floating point values are viewed as `double`.

Automatic Promotion

Example of potential problem:

```
short a, b, c;
a = 1 ;
b = 2 ;
c = a + b ; //compiler error
```

a and b are automatically promoted to integers.

Example of potential solutions:

- Declare `c` as an `int` type in the original declaration:

```
int c;
```

- Type cast the `(a+b)` result in the assignment line:

```
c = (short)(a+b);
```

Using a `long`

```
1 public class Person {
2
3   public int ageYears = 32;
4
4   public void calculateAge() {
5
6     int ageDays = ageYears * 365;
7     long ageSeconds = ageYears * 365 * 24L * 60 * 60;
8
9     System.out.println("You are " + ageDays + " days old.");
10    System.out.println("You are " + ageSeconds + " seconds old.");
11
12   } // end of calculateAge method
13  } // end of class
```

Using the L to indicate a long will result in the compiler recognizing the total result as a long.

Using Floating Points

Example of potential problem:

Expressions are automatically promoted to floating points.

```
int num1 = 1 + 2 + 3 + 4.0;                    //compiler error
int num2 = (1 + 2 + 3 + 4) * 1.0;              //compiler error
```

Example of potential solutions:

- Declare `num1` and `num2` as `double` types:

```
double num1 = 1 + 2 + 3 + 4.0;                 //10.0
double num2 = (1 + 2 + 3 + 4) * 1.0;           //10.0
```

- Type cast `num1` and `num2` as `int` types in the assignment line:

```
int num1 = (int)(1 + 2 + 3 + 4.0);             //10
int num2 = (int)((1 + 2 + 3 + 4) * 1.0);       //10
```

Floating Point Data Types and Assignment

- Example of potential problem:

```
float float1 = 27.9;   //compiler error
```

- Example of potential solutions:
  - The `F` notifies the compiler that 27.9 is a `float` value:

```
float float1 = 27.9F;
```

  - `27.9` is cast to a `float` type:

```
float float1 = (float) 27.9;
```

Quiz

Which statements are true?

a. There are eight primitive types built in to the Java programming language.

b. `byte`, `short`, `char`, and `long` are the four integral primitive data types in the Java programming language.

c. A `boolean` type variable holds `true`, `false`, and `nil`.

d. `short Long = 10;` is a valid statement that adheres to the variable declaration and initialization syntax.

Exercise 7-3: Declare a `long`, `float`, and `char`

1. Open the project **`Practice_07-3`** in NetBeans.
2. Declare a `long`, using the `L` to indicate a long value. Make it a very large number (in the billions).
3. Declare and initialize a `float` and a `char`
4. Print the `long` variable with a suitable label.
5. Assign the `long` to the `int` variable. Correct the syntax error by casting the `long` as an `int`.
6. Print the `int` variable. Note the change in value when you run it.

Summary

In this lesson, you should have learned how to:

- Describe the `String` class and use some of the methods of the `String` class
- Use the JDK API documentation to search for and learn how to use a class
- Use the `StringBuilder` class to manipulate string data
- Create a constant by using the final keyword in the variable declaration
- Describe how the Java compiler can use promotion or casting to interpret expressions and avoid a compiler error

Practices Overview

- 7-1: Manipulating Text