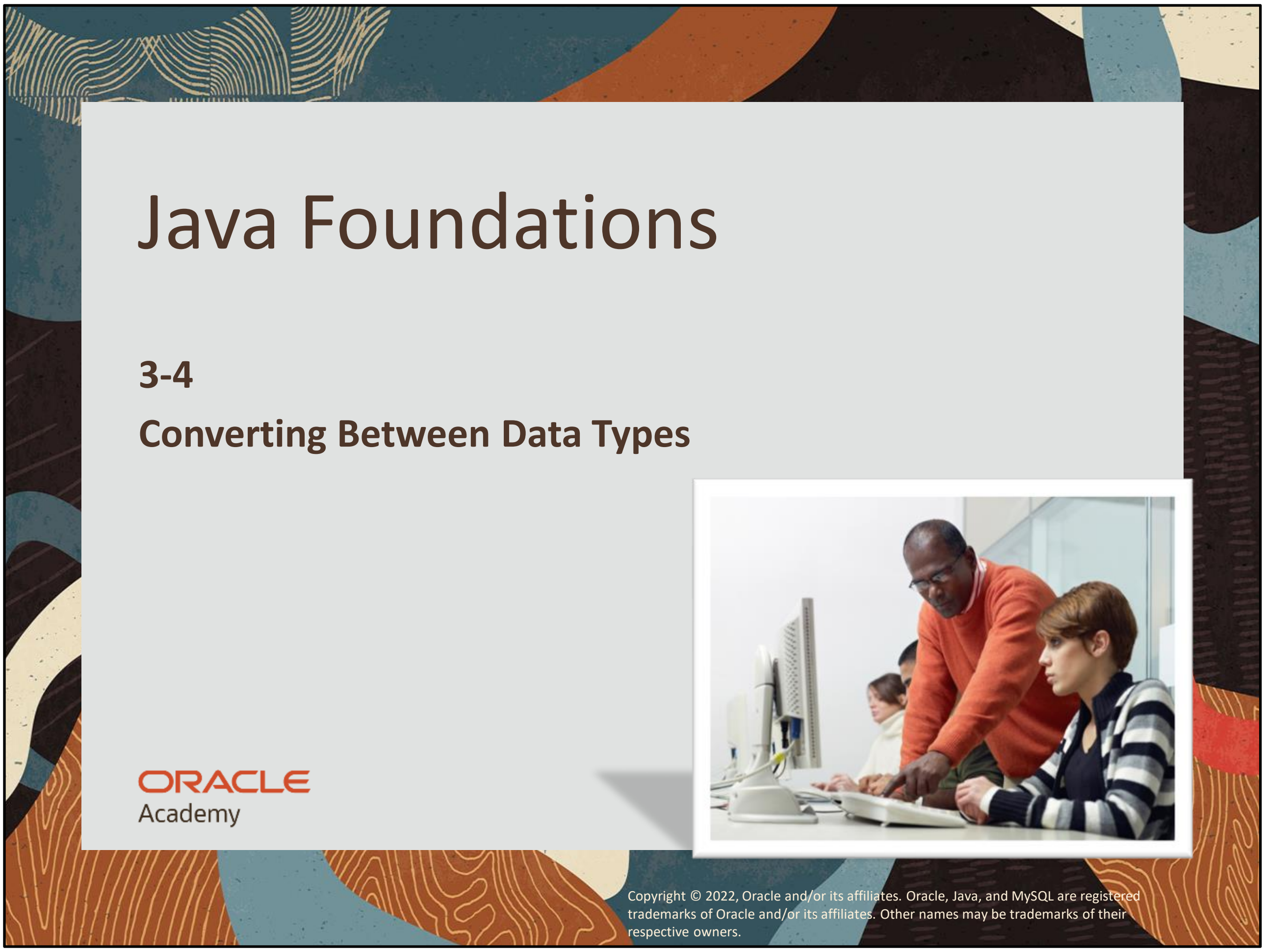
ORACLE

Academy



# Java Foundations

## 3-4
## Converting Between Data Types

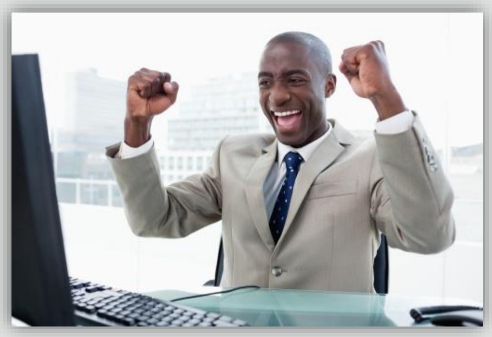
ORACLE
Academy

# Objectives

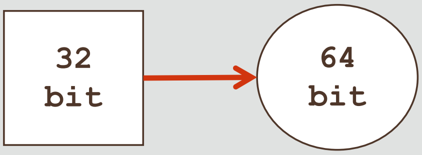
- This lesson covers the following objectives:
  - Take advantage of automatic promotion
    - And when to be cautious with promotions
  - Cast variables to other data types
    - And when to be cautious with casting
  - Parse Strings as numeric values

# Congratulations!

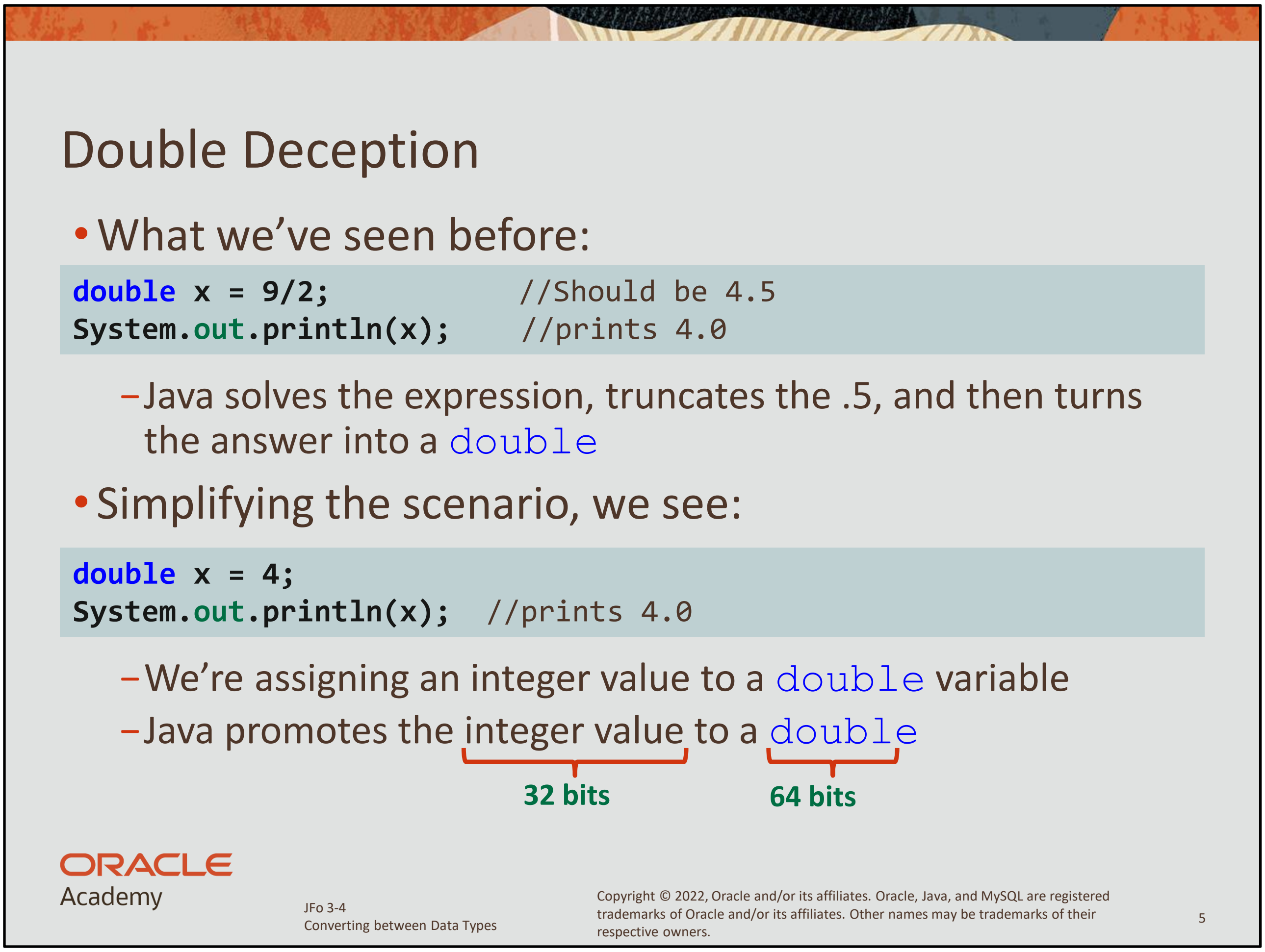
- Congratulations on making it this far in the course!
- A promotion is coming your way!



- Your promotion:



32 bit → 64 bit



## Double Deception

- What we’ve seen before:

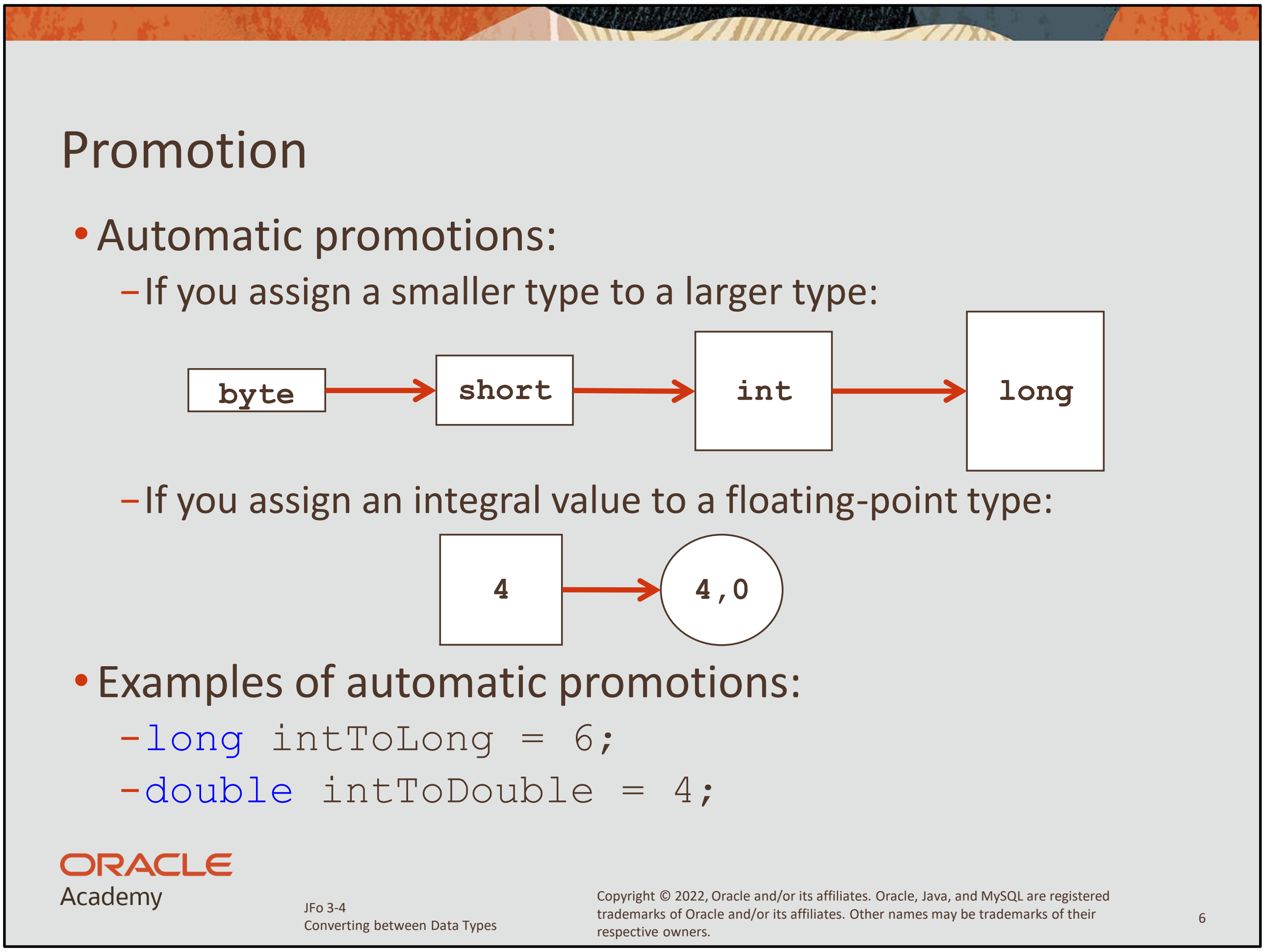
```
double x = 9/2;            //Should be 4.5
System.out.println(x);     //prints 4.0
```

  - Java solves the expression, truncates the .5, and then turns the answer into a `double`

- Simplifying the scenario, we see:

```
double x = 4;
System.out.println(x);   //prints 4.0
```

  - We’re assigning an integer value to a `double` variable
  - Java promotes the $\underbrace{\text{integer value}}_{\text{32 bits}}$ to a $\underbrace{\texttt{double}}_{\text{64 bits}}$



## Promotion

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

byte → short → int → long

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

4 → 4,0

- Examples of automatic promotions:
  - `long intToLong = 6;`
  - `double intToDouble = 4;`

ORACLE
Academy

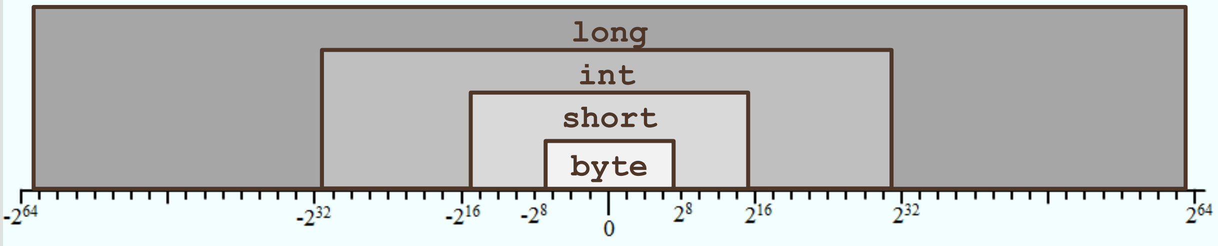
JFo 3-4
Converting between Data Types

In some circumstances, the compiler changes the type of a variable to a type that supports a larger size value. This action is referred to as a promotion. Some promotions are done automatically by the compiler if data would not be lost by doing so.
These promotions include:

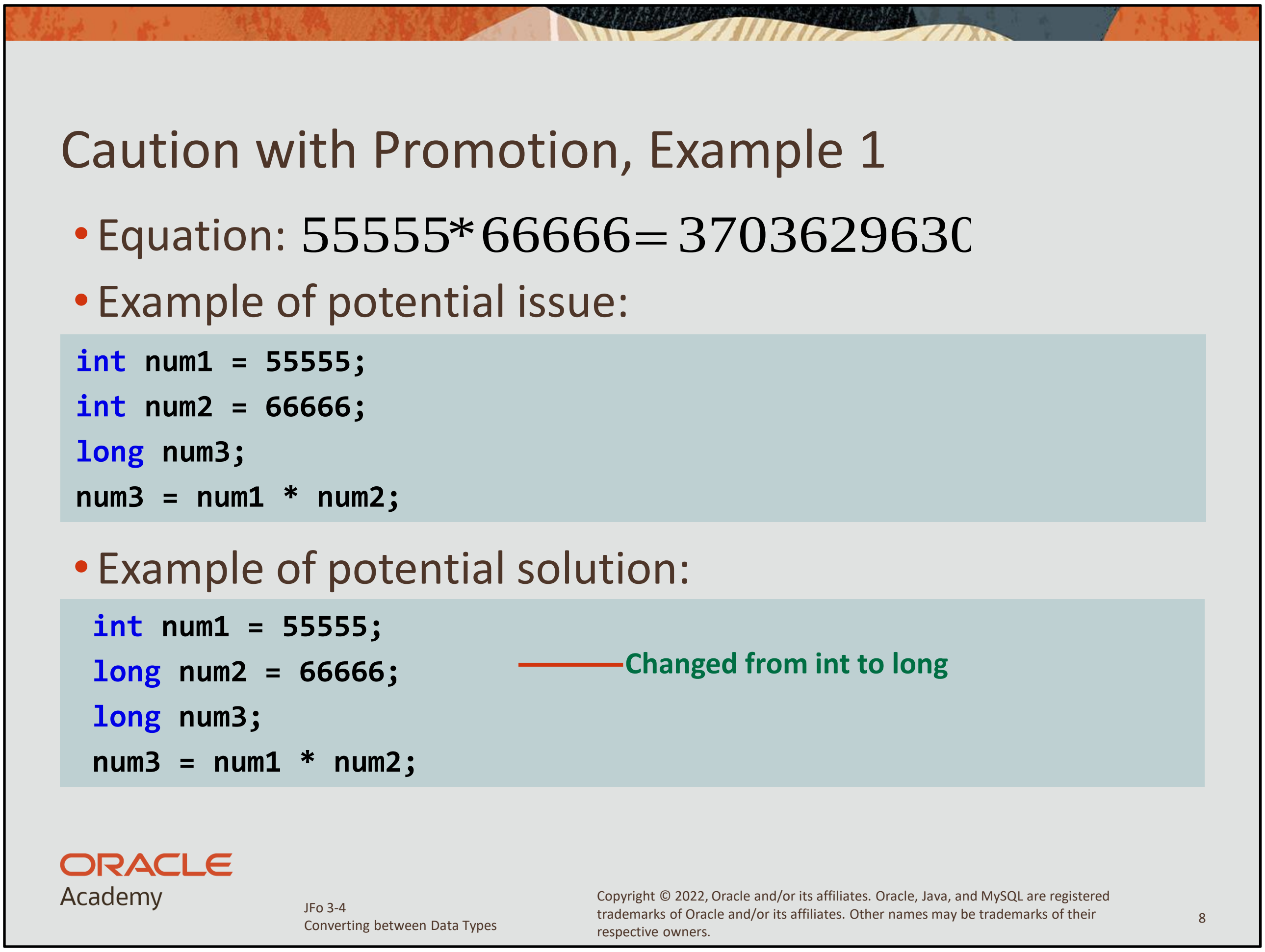
If you assign a smaller type (on the right of the equal sign) to a larger type (on the left of the equal sign)

If you assign an integral type to a floating point type (because there are no decimal places to lose)

## Why Does Promotion Work?



- A byte could be -128 to 127
- All possible byte values can be contained in a short
- All possible short values can be contained in an int
- All possible int values can be contained in a long
- All possible int values can be contained in a double without losing precision



## Caution with Promotion, Example 1

- Equation: $55555*66666=3703629630$
- Example of potential issue:

```
int num1 = 55555;
int num2 = 66666;
long num3;
num3 = num1 * num2;
```

- Example of potential solution:

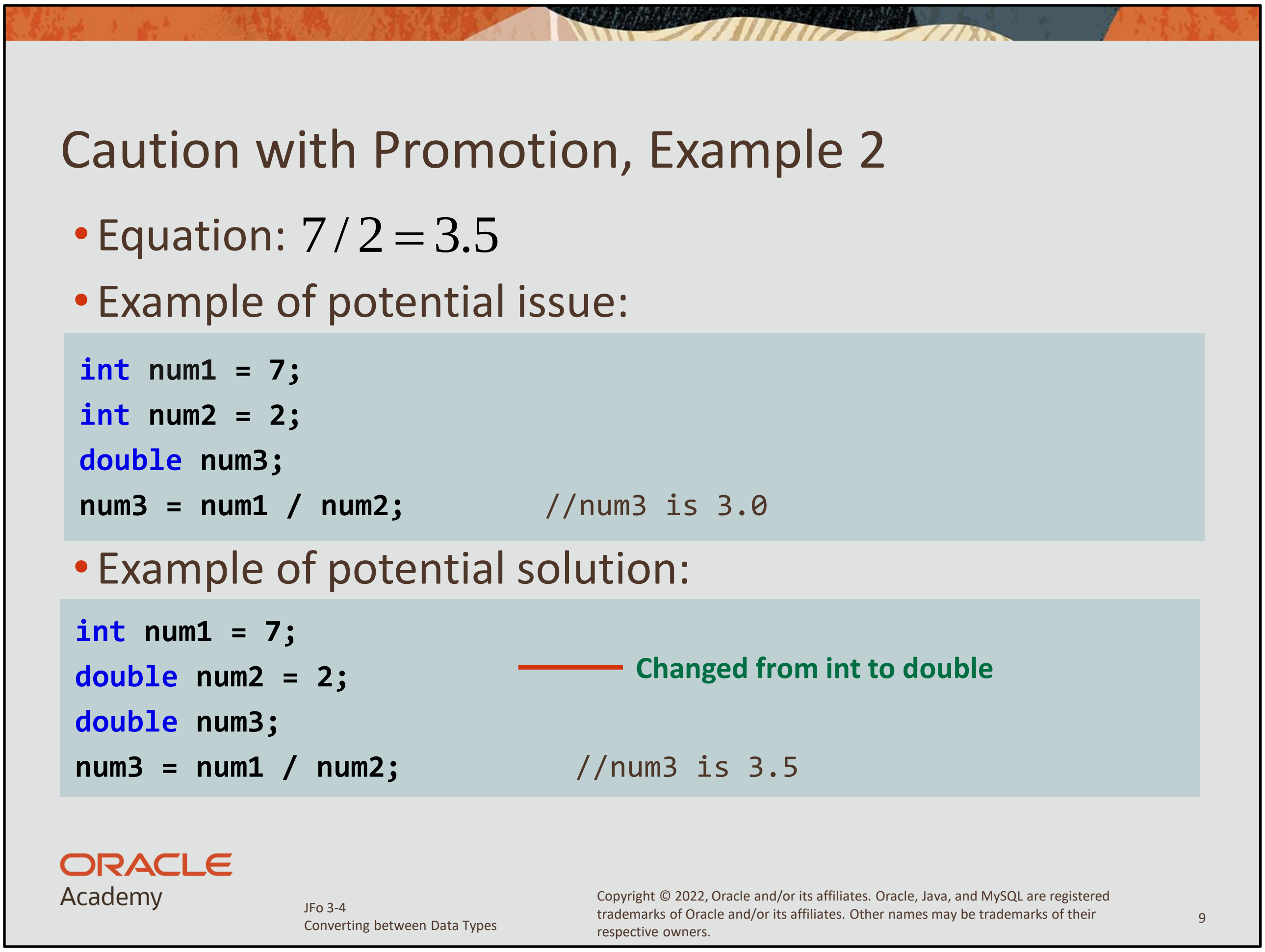
```
int num1 = 55555;
long num2 = 66666;
long num3;
num3 = num1 * num2;
```

Changed from int to long

ORACLE Academy

Prior to being assigned to a variable, the result of an equation is placed in a temporary location in memory. The location’s size is always equal to the size of an int type or the size of the largest data type used in the expression or statement. For example, if your equation multiplies two int types, the container size will be an int type in size, or 32 bits.

If the two values that you multiply yield a value that is beyond the scope of an int type, (such as 55555 * 66666 = 3,703,629,630, which is too big to fit in an int type), the int value must be truncated to fit the result into the temporary location in memory. This calculation ultimately yields an incorrect answer because the variable for your answer receives a truncated value (regardless of the type used for your answer). To solve this problem, set at least one of the variables in your equation to the long type to ensure the largest possible temporary container size.



## Caution with Promotion, Example 2

- Equation: $7/2=3.5$
- Example of potential issue:

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

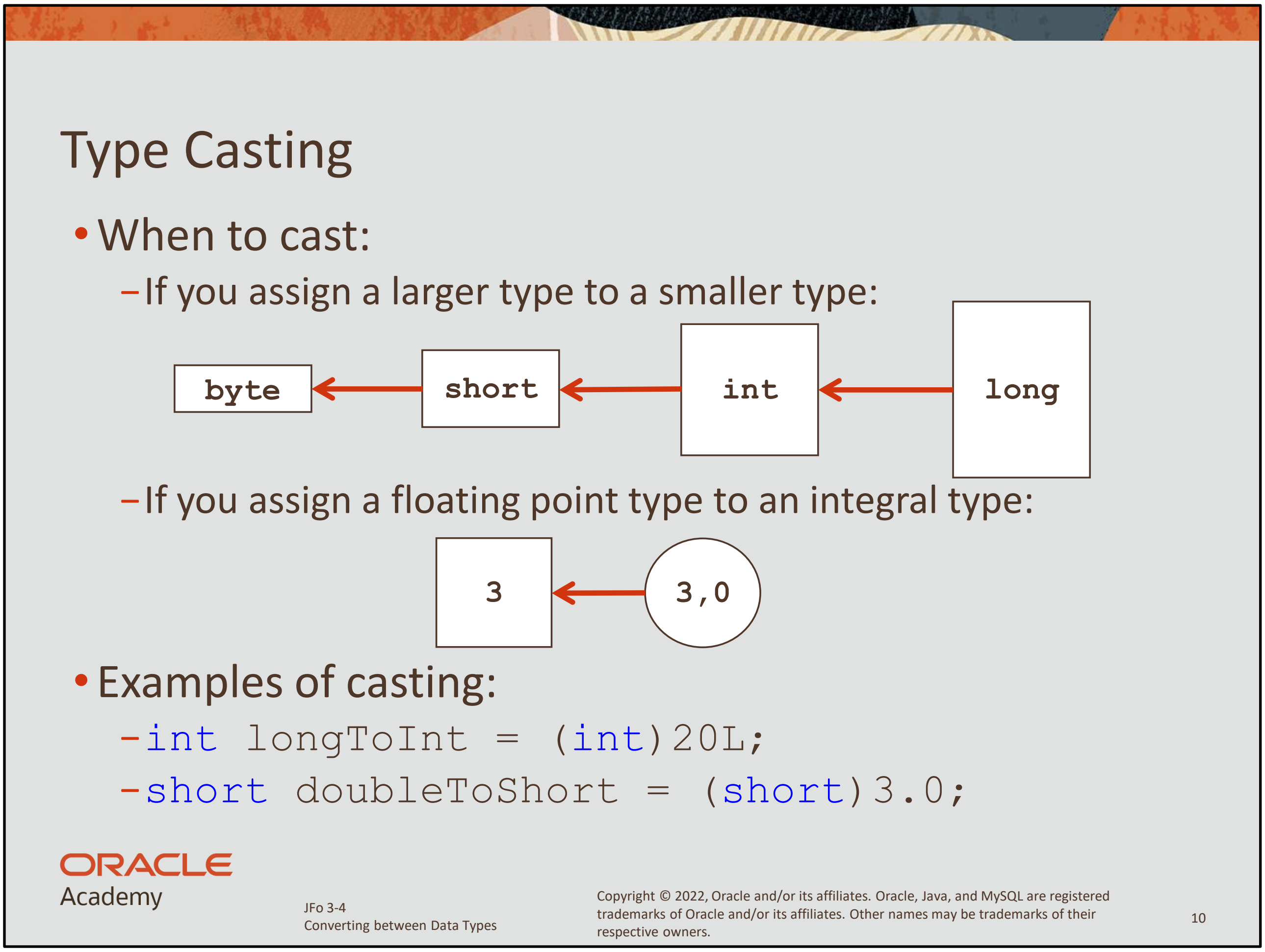
- Example of potential solution:

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

Changed from int to double

ORACLE
Academy

Integer division may lead to a loss of decimal precision. The same issue occurs with other data types. Prior to being assigned to a variable, the result of an equation is placed in a temporary location in memory. The location's size is always equal to the size of the largest data type used in the expression or statement. For example, if your equation divides two int types, the container size will be an int type in size, or 32 bits.



## Type Casting

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

byte
short
int
long

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

3
3,0

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

Type casting lowers the range of a value, quite literally chopping it down to a smaller size, by changing the type of the value (for example, by converting a long value to an int value). You do this to use methods that accept only certain types as arguments. Then, you can assign values to a variable of a smaller data type or you can save memory.

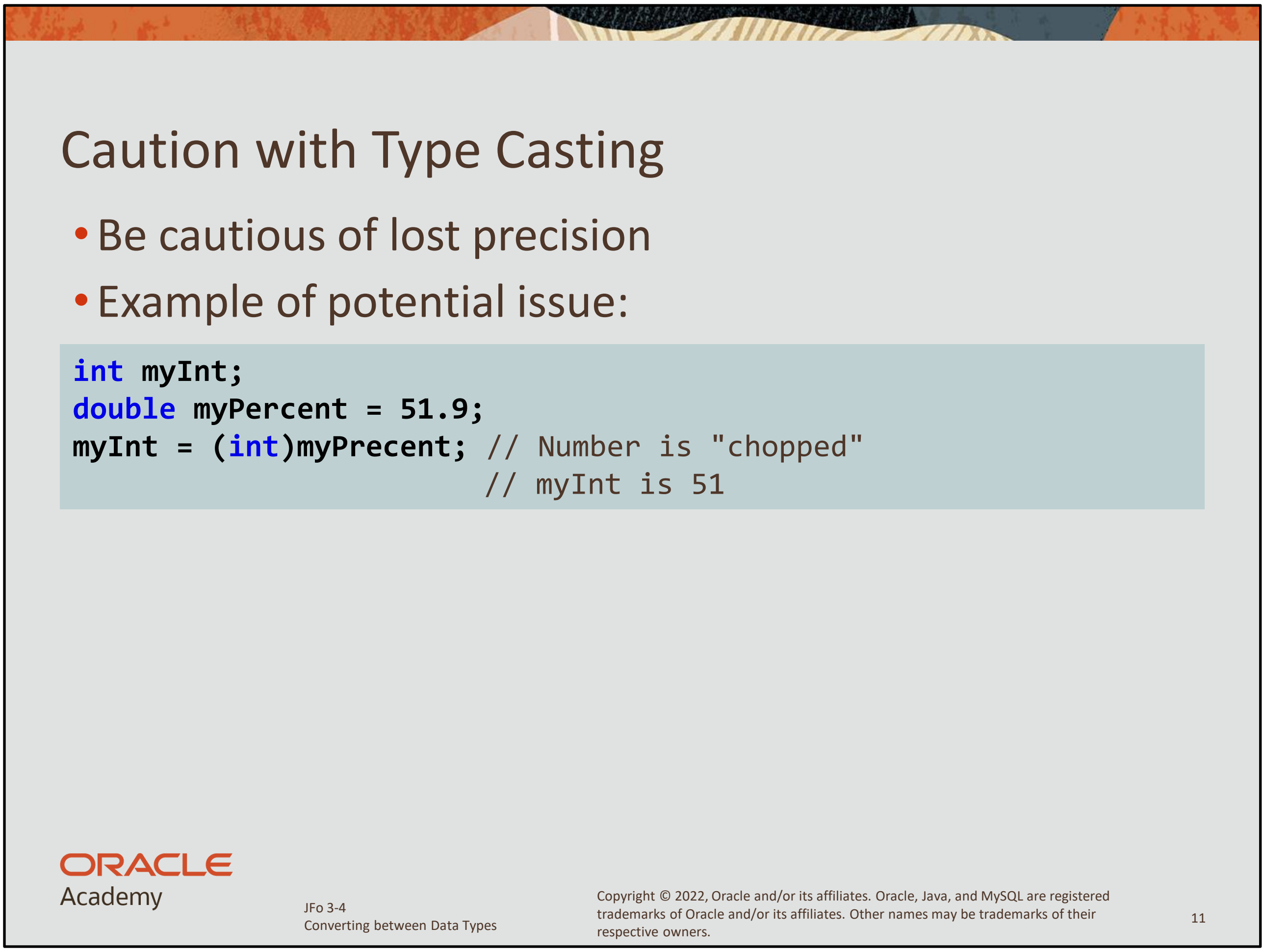
The syntax for type casting a value is **identifier = (target_type) value.**

In the syntax:

identifier is the name that you assign to the variable.

value is the value you want to assign to the identifier.

(target_type) is the type to which you want to type cast the value. Notice that the target_type must be enclosed in parentheses.

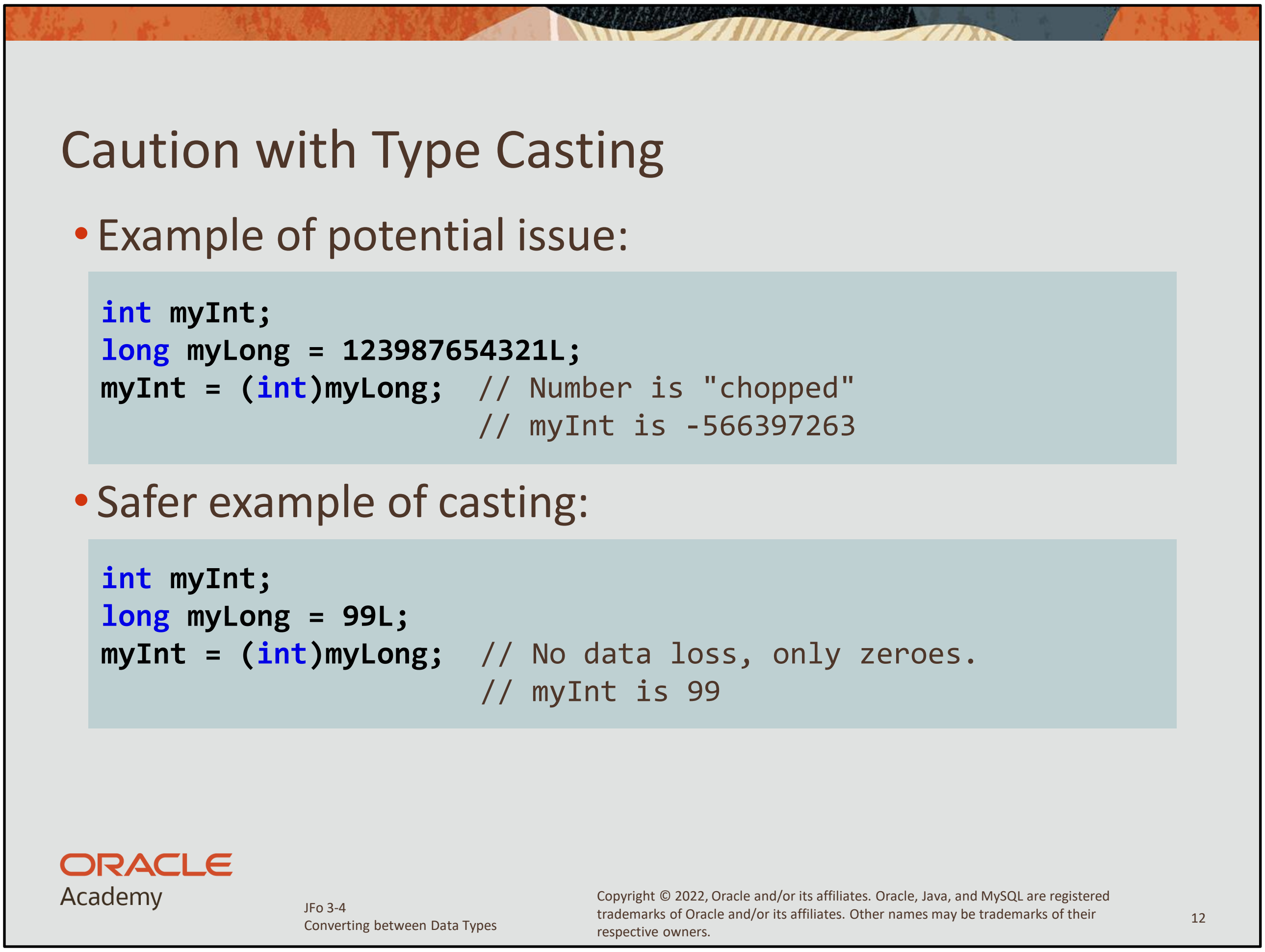


## Caution with Type Casting

- Be cautious of lost precision
- Example of potential issue:

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

If you type cast a float or double value with a fractional part to an integral type such as an int, all decimal values are lost. However, this method of type casting is sometimes useful if you want to truncate the number down to the whole number (for example, 51.9 becomes 51).



## 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
```

ORACLE
Academy

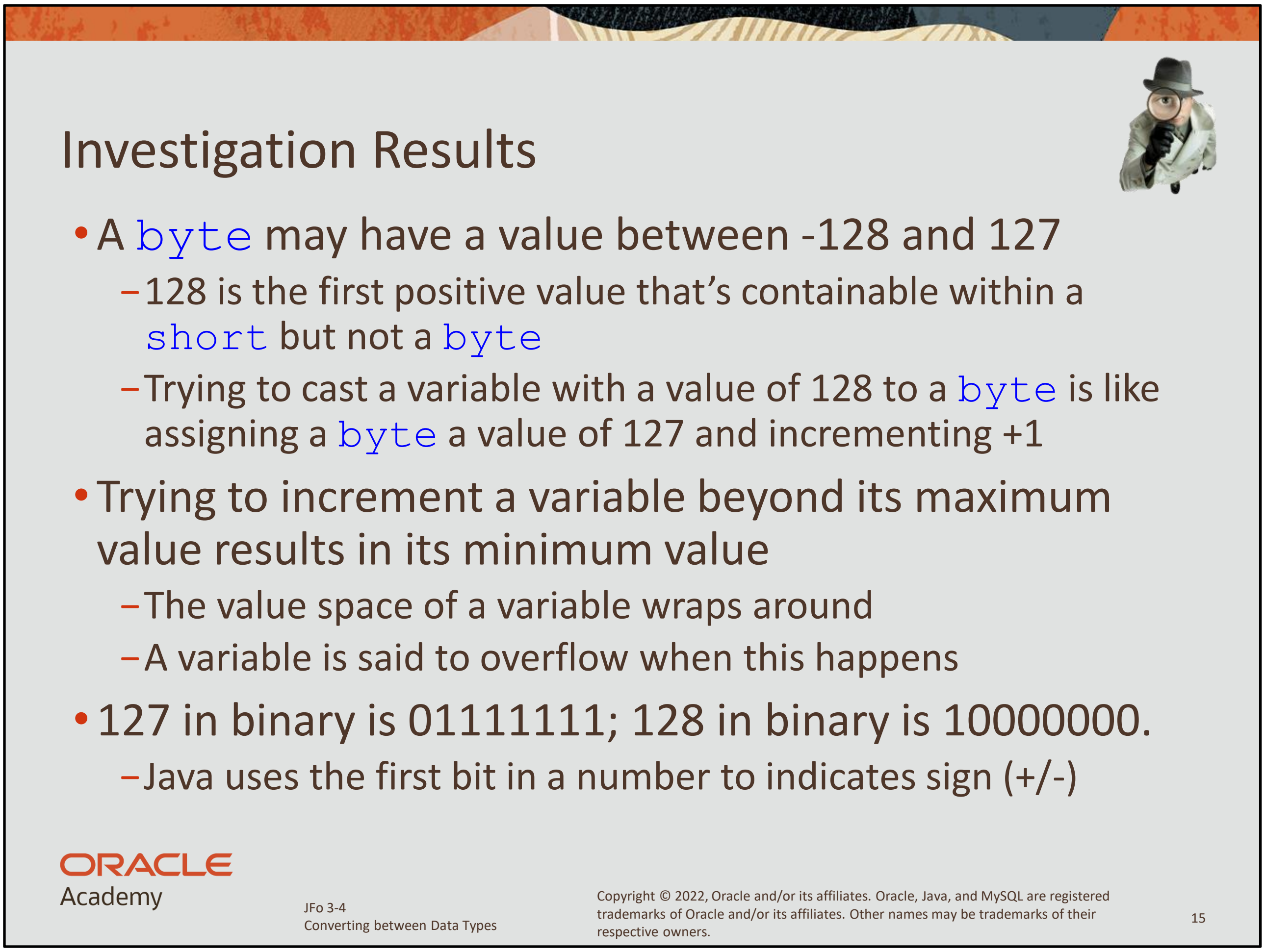
The loss of precision with casting can sometimes lead to situations where numbers are truncated, leading to errors in calculations.

# Chopping an Integral

- The examples we’ve seen raise a few questions:
  - What does it mean to “chop” an integral?
  - Why are we getting negative values?
- It’s time to launch another investigation with …
  - Casting
  - Math

# Exercise 1

- Create a new project and add the `Casting01.java` file to the project
- Declare and initialize a `byte` with a value of 128:
  - Observe NetBeans' complaint
  - Comment out this problematic line
- Declare and initialize a `short` with a value of 128:
  - Create a print statement that casts this `short` to a `byte`
- Declare and initialize a `byte` with a value of 127
  - Add 1 to this variable and print it
  - Add 1 to this variable again and print it again



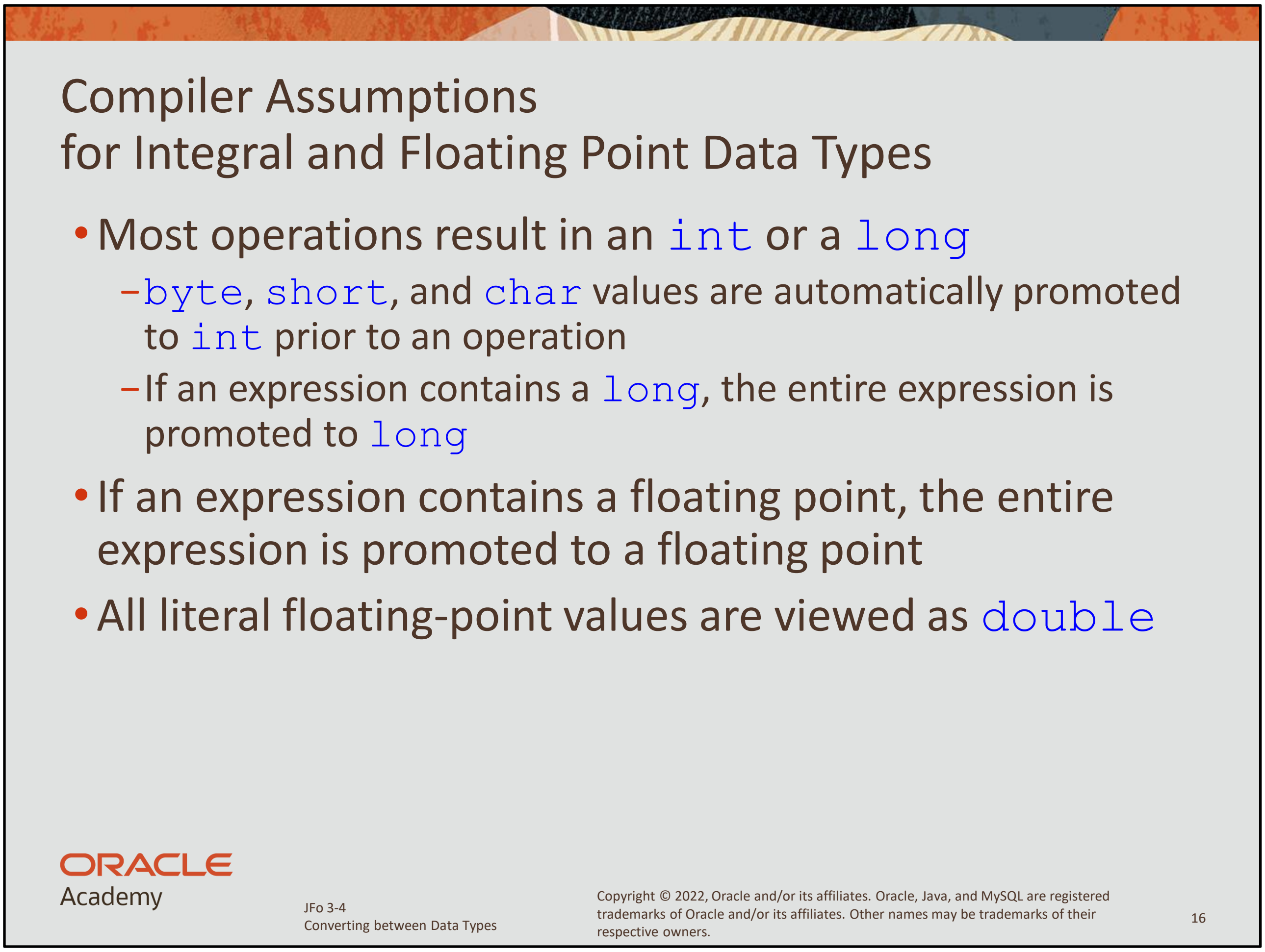
## Investigation Results

- A `byte` may have a value between -128 and 127
  - 128 is the first positive value that’s containable within a `short` but not a `byte`
  - Trying to cast a variable with a value of 128 to a `byte` is like assigning a `byte` a value of 127 and incrementing +1
- Trying to increment a variable beyond its maximum value results in its minimum value
  - The value space of a variable wraps around
  - A variable is said to overflow when this happens
- 127 in binary is 01111111; 128 in binary is 10000000.
  - Java uses the first bit in a number to indicates sign (+/-)

ORACLE
Academy

JFo 3-4
Converting between Data Types

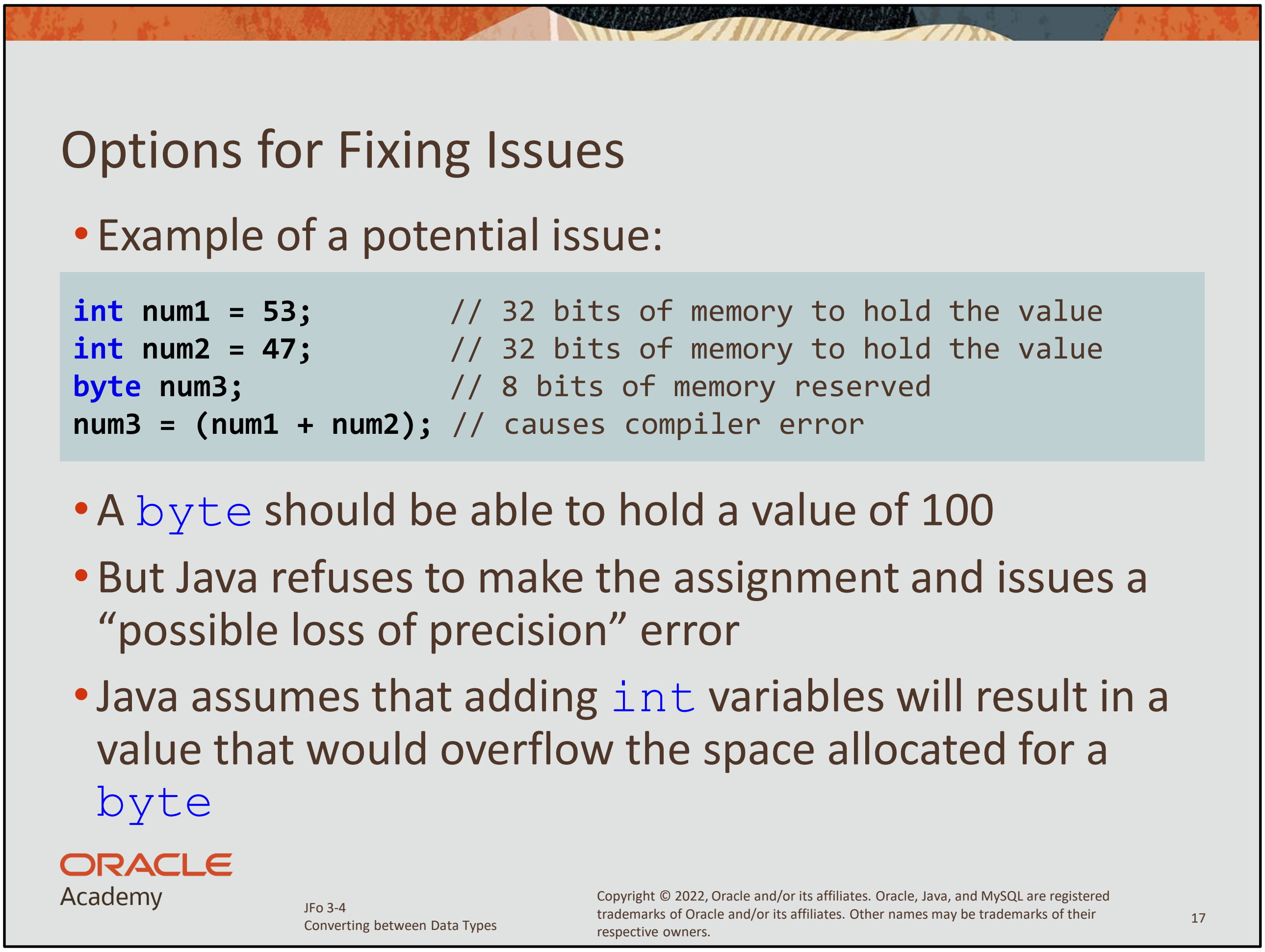
Consider this background information. It isn’t necessary to remember these facts to finish problem sets. It’s more important to understand how to promote and cast.



## Compiler Assumptions for Integral and Floating Point Data Types

- Most operations result in an `int` or a `long`
  - `byte`, `short`, and `char` 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`

The Java technology compiler makes certain assumptions when it evaluates expressions. You must understand these assumptions to make the appropriate type casts or other accommodations. The next few slides give examples.

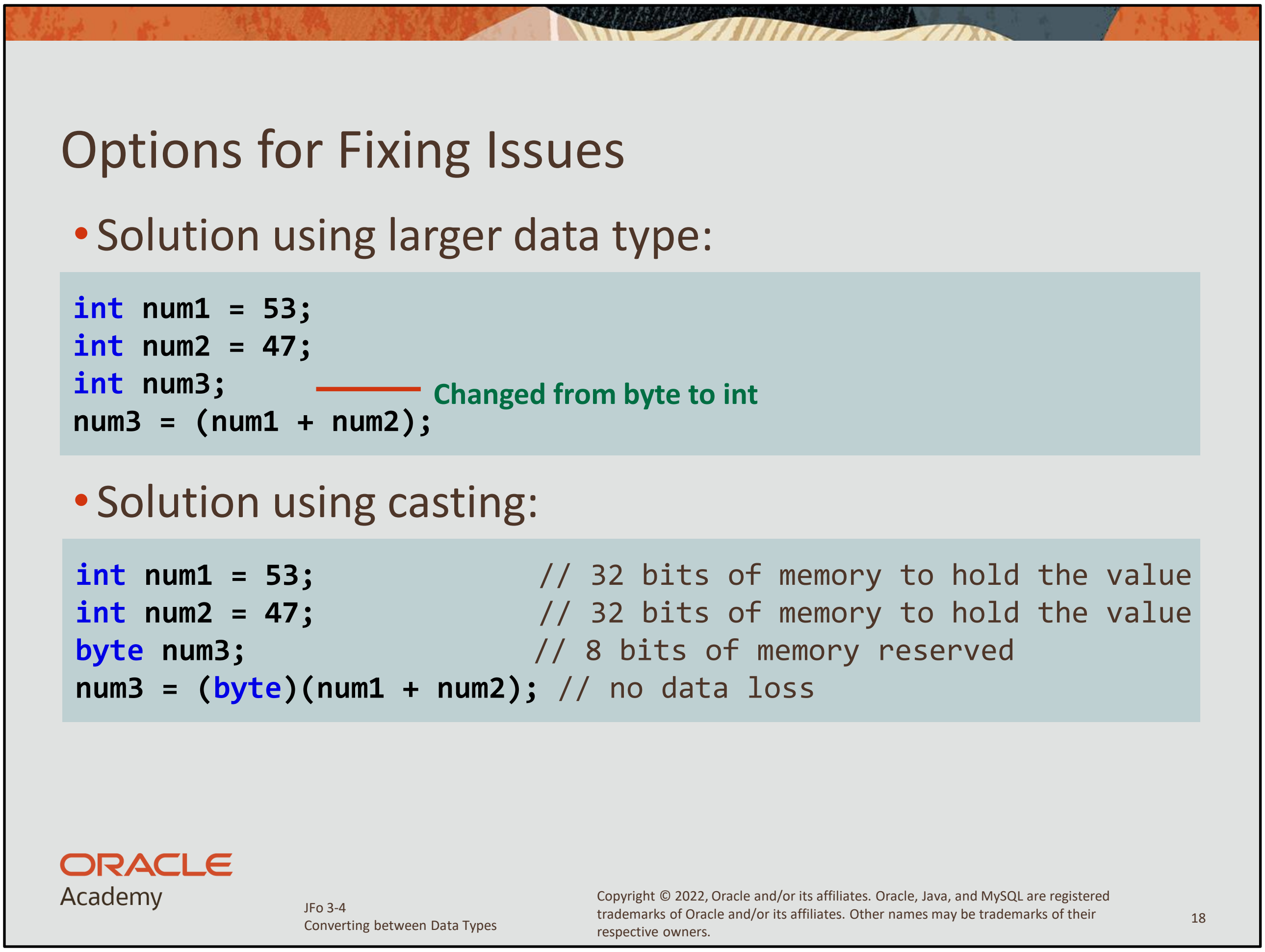


## Options for Fixing Issues

- Example of a 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
```

- A `byte` should be able to hold a value of 100
- But Java refuses to make the assignment and issues a “possible loss of precision” error
- Java assumes that adding `int` variables will result in a value that would overflow the space allocated for a `byte`



## Options for Fixing Issues

- Solution using larger data type:

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

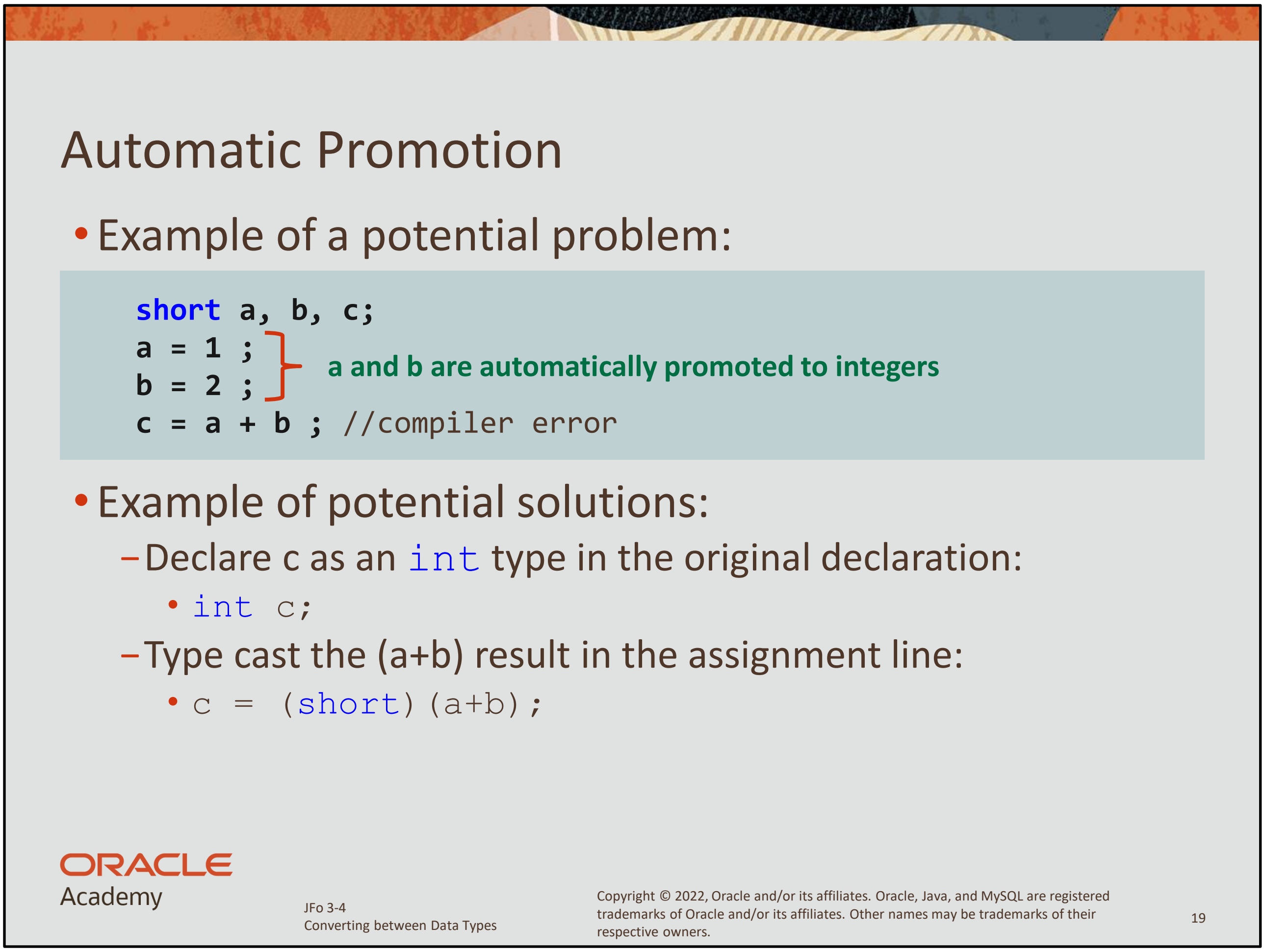
Changed from byte to int

- 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
```

ORACLE Academy

To fix a “possible loss of precision” error, you can either (a) declare the variable on the left side (num3) to be a larger data type such as an int, or (b) type cast the right-side data type down to match the left-side data type.



## Automatic Promotion

- Example of a potential problem:

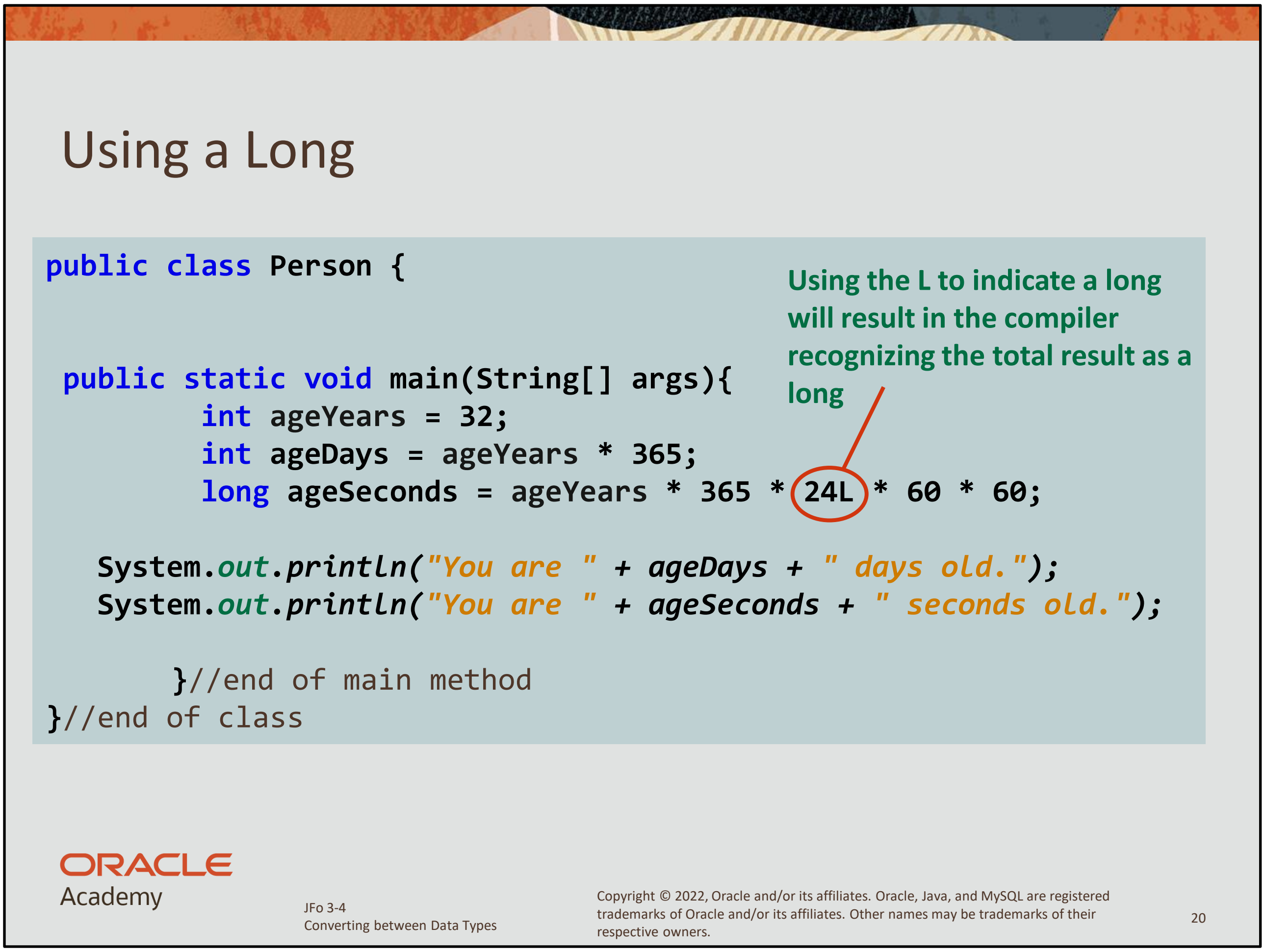
```
short a, b, c;
a = 1 ;
b = 2 ;
c = a + b ; //compiler error
```

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);`

ORACLE
Academy

In the slide example, an error occurs because two of the three operands (a and b) are automatically promoted from a short type to an int type before they’re added.
In the last line, the values of a and b are converted to int types, and the converted values are added to give an int result. Then the assignment operator (=) attempts to assign the int result to the short variable (c). However, this assignment is illegal and causes a compiler error.



# Using a Long

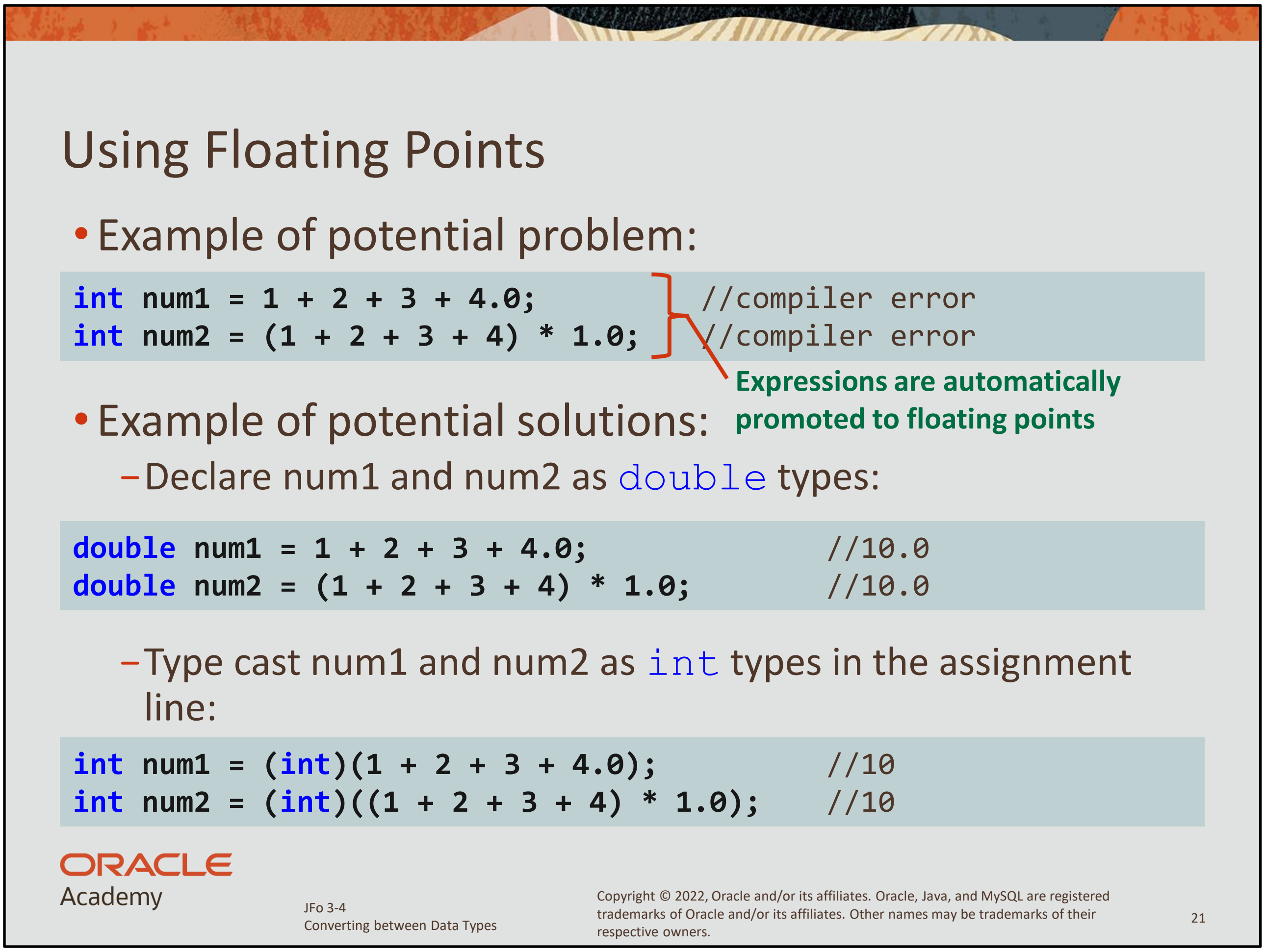
```
public class Person {


 public static void main(String[] args){
         int ageYears = 32;
         int ageDays = ageYears * 365;
         long ageSeconds = ageYears * 365 * 24L * 60 * 60;

   System.out.println("You are " + ageDays + " days old.");
   System.out.println("You are " + ageSeconds + " seconds old.");

       }//end of main method
}//end of class
```

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

ORACLE Academy

JFo 3-4
Converting between Data Types

The code example uses principles from this section to calculate a person’s age in days and seconds. Because the ageSeconds variable is declared as a long, one of the literal values used as operands in the assigned expression must be initialized as a long value (‘L’) so that the compiler will allow the assignment.



## Using Floating Points

- Example of potential problem:

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

Expressions are automatically promoted to floating points

- Example of potential solutions:
  - Declare num1 and num2 as `double` types:

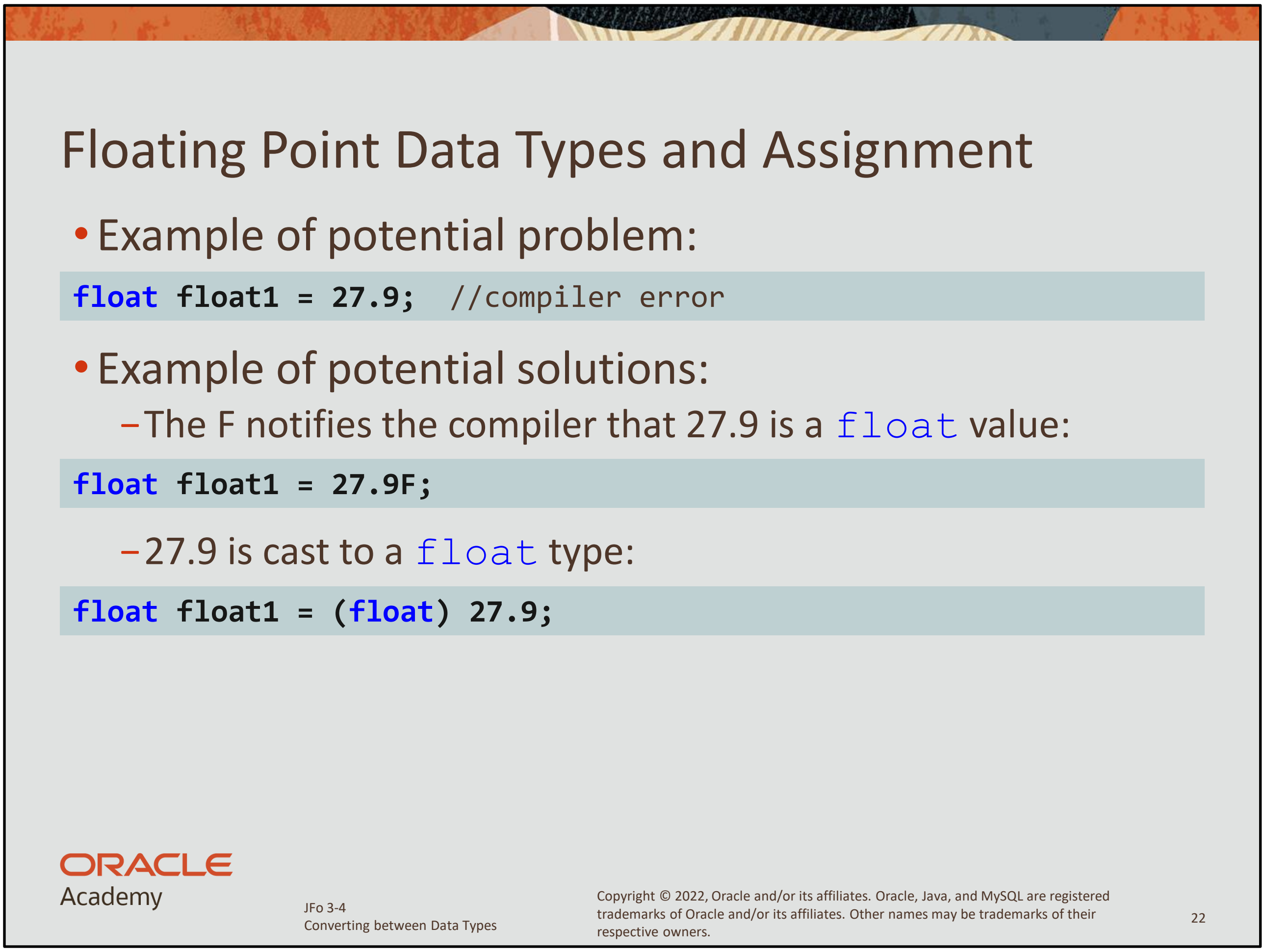
```
double num1 = 1 + 2 + 3 + 4.0;            //10.0
double num2 = (1 + 2 + 3 + 4) * 1.0;      //10.0
```

  - 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
```

ORACLE Academy

If an expression contains a floating point, the entire expression is promoted to a floating point.



# 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:

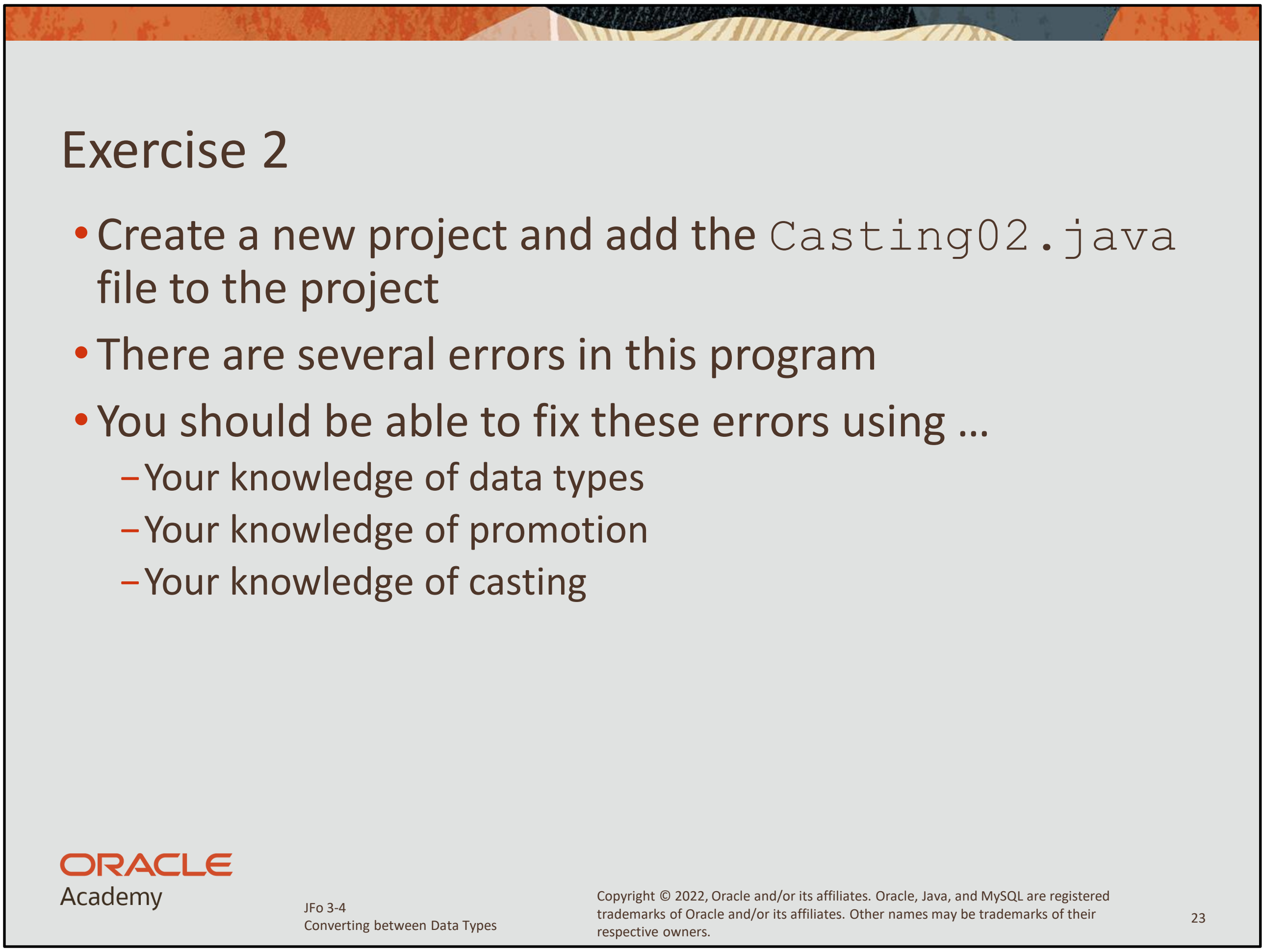
```
float float1 = 27.9F;
```

  - 27.9 is cast to a `float` type:

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

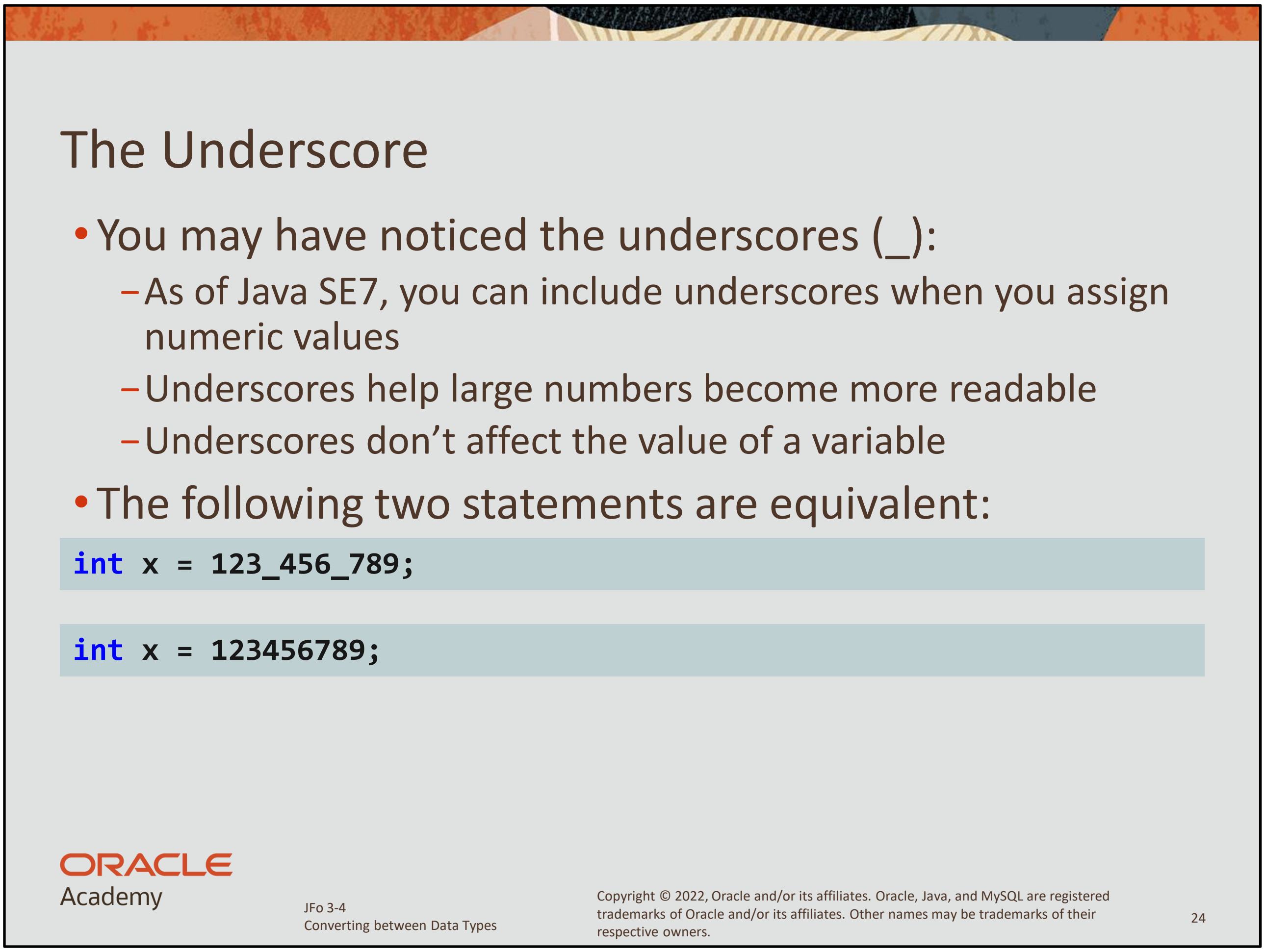
ORACLE
Academy

Just as integral types default to int under some circumstances, values assigned to floating point types always default to a double type, unless you specifically state that the value is a float type. This is done by placing a capital F after a number value.  Otherwise 27.9 is assumed to be a double type, and a compiler error occurs because a double type value cannot fit into a float variable.



## Exercise 2

- Create a new project and add the `Casting02.java` file to the project
- There are several errors in this program
- You should be able to fix these errors using …
  - Your knowledge of data types
  - Your knowledge of promotion
  - Your knowledge of casting



## The Underscore

- You may have noticed the underscores (_):
  - As of Java SE7, you can include underscores when you assign numeric values
  - Underscores help large numbers become more readable
  - Underscores don’t affect the value of a variable
- The following two statements are equivalent:

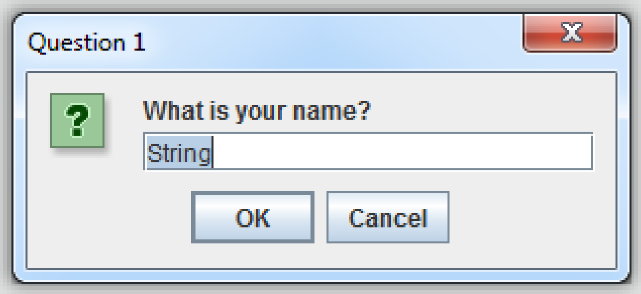
```
int x = 123_456_789;
```

```
int x = 123456789;
```

ORACLE
Academy

# Converting Strings to Numeric Data

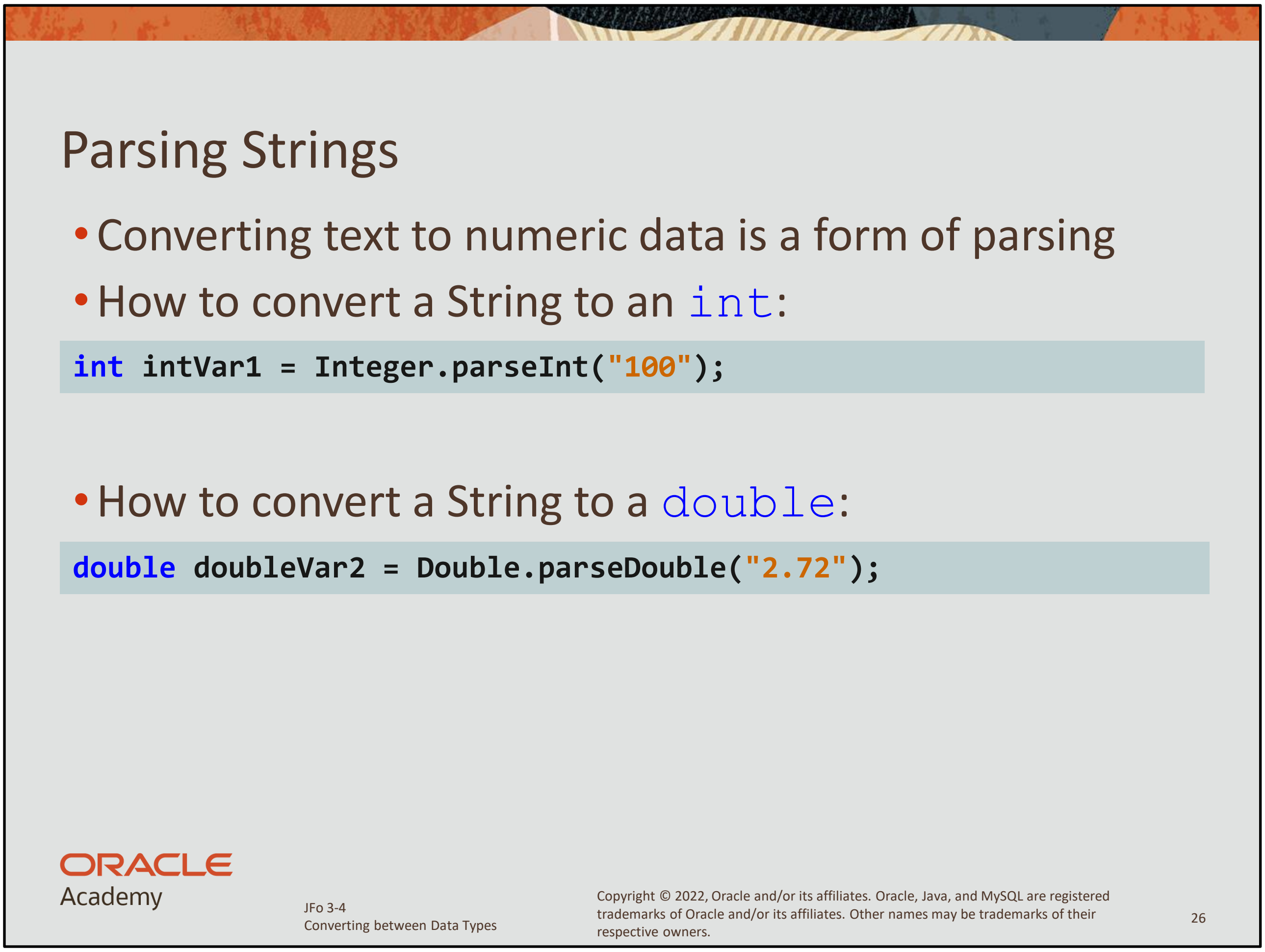
- When you invite a user to type in a dialog box …
  - They can type whatever text they want
  - This text is best represented by a String
- But sometimes you'll need to do math with user inputs
  - If you design a program that accepts text input, you may have to convert the String to numeric data types



Question 1

What is your name?

String



## Parsing Strings

- Converting text to numeric data is a form of parsing
- How to convert a String to an `int`:

```
int intVar1 = Integer.parseInt("100");
```

- How to convert a String to a `double`:

```
double doubleVar2 = Double.parseDouble("2.72");
```

ORACLE
Academy

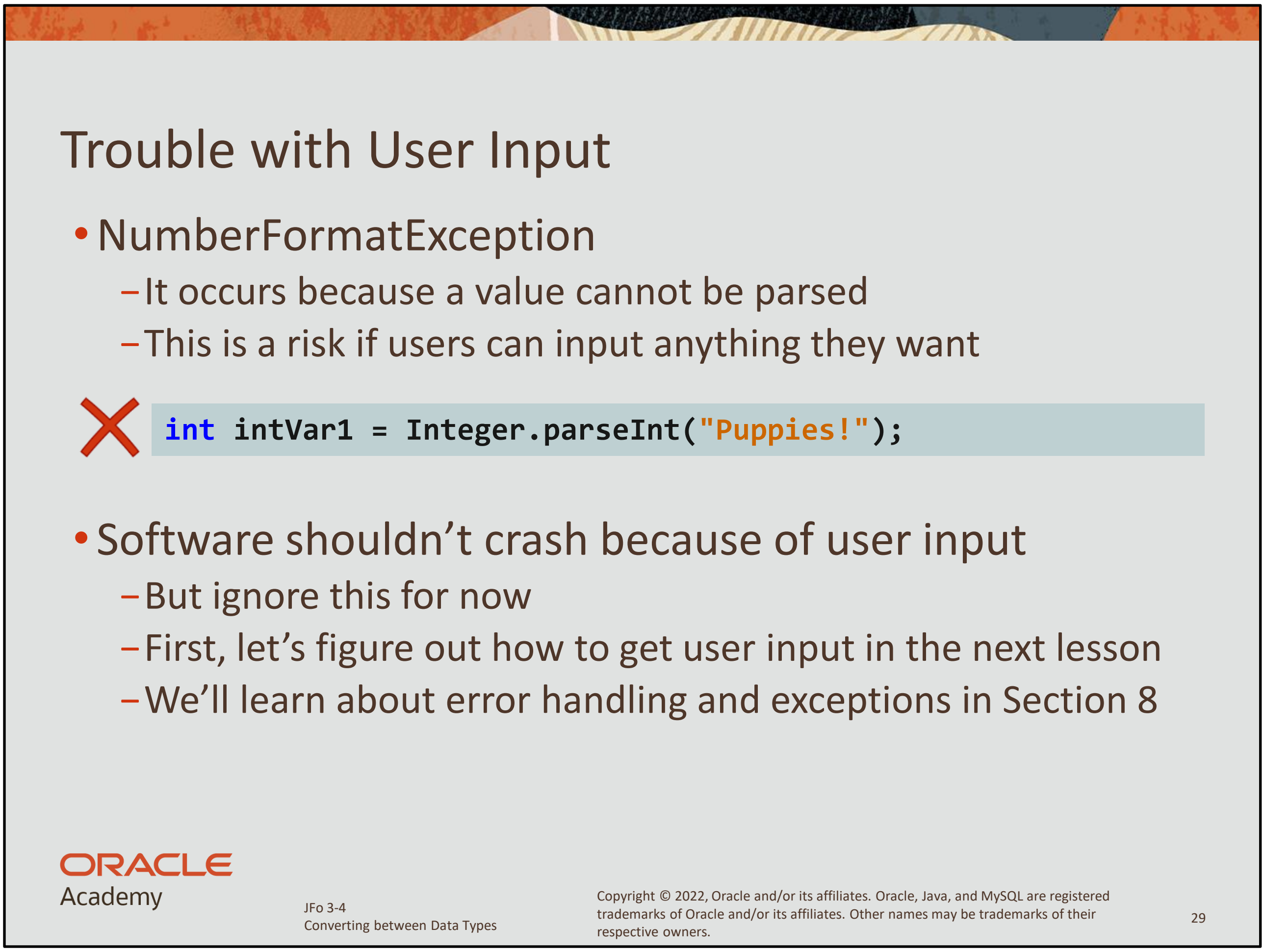
# Exercise 3, Part 1

- Create a new project and add `Parsing01.java` file to the project
- Declare and initialize 3 Strings with the following data:

| String Variable | Description | Example Values |
|---|---|---|
| shirtPrice | Text to be converted to an `int`: | "15" |
| taxRate | Text to be converted to a `double`: | "0.05" |
| gibberish | Gibberish | "887ds7nds87dsfs" |

# Exercise 3, Part 2

- Parse and multiply shirtPrice*taxRate to find the tax
  - Print this value
- Try to parse taxRate as an `int`
  - Observe the error message
- Try to parse gibberish as an `int`
  - Observe the error message



# Trouble with User Input

- NumberFormatException
  - It occurs because a value cannot be parsed
  - This is a risk if users can input anything they want

✗

```
int intVar1 = Integer.parseInt("Puppies!");
```

- Software shouldn’t crash because of user input
  - But ignore this for now
  - First, let’s figure out how to get user input in the next lesson
  - We’ll learn about error handling and exceptions in Section 8

ORACLE
Academy

# Summary

- In this lesson, you should have learned how to:
  - Take advantage of automatic promotion
    - And when to be cautious with promotions
  - Cast variables to other data types
    - And when to be cautious with casting
  - Parse Strings as numeric values

ORACLE
Academy