ORACLE
Academy

Oracle Academy
Java for AP Computer Science A

3-4

Converting Between Data Types

Objectives

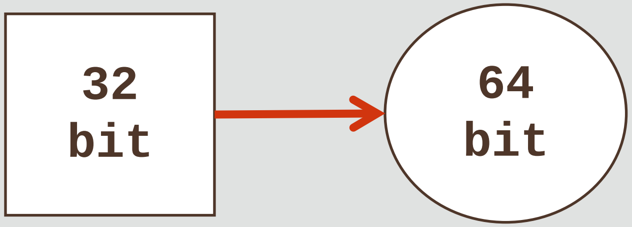
- This lesson covers the following objectives:
  - Take advantage of automatic promotion
    - And when to be cautious with promotions
  - Cast variables to other data types
    - And when to be cautious with casting
  - Parse Strings as numeric values

Congratulations!

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

Wow!

- Your promotion:



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`

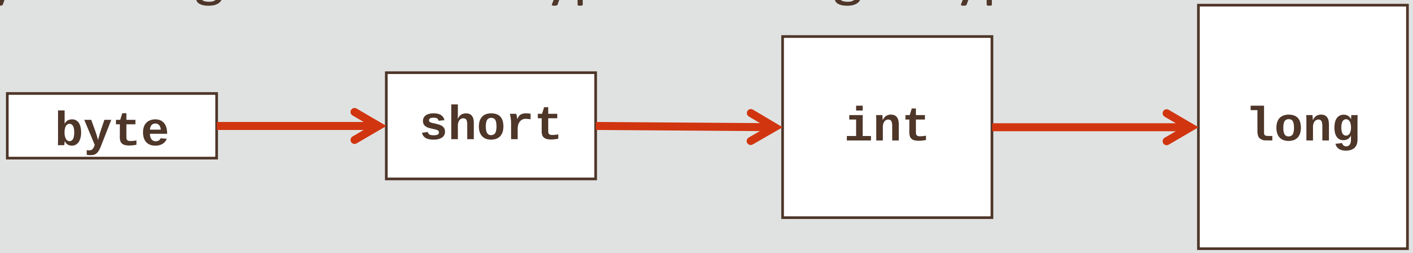
- Simplifying the scenario, we see:

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

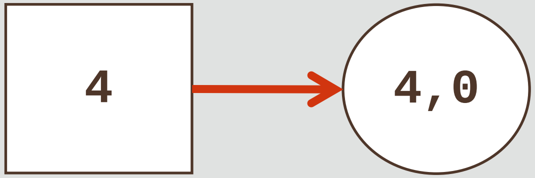
  - We're assigning an integer value to a `double` variable
  - Java promotes the $\underbrace{\text{integer value}}_{\text{32 bits}}$ to a $\underbrace{\texttt{double}}_{\text{64 bits}}$

Promotion

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



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



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

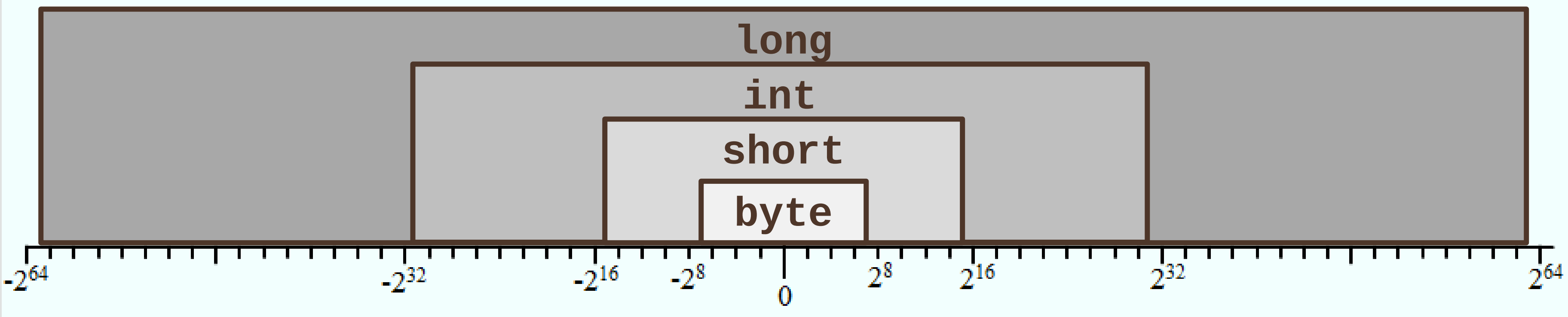
Determining Numeric Data Type Range

- Java constants can be used to determine the range for numeric data types, and to ensure that input data will not cause errors:

```
int intmin = Integer.MIN_VALUE;
System.out.println(intmin);  //output is  -2147483648
```

```
int intmax = Integer.MAX_VALUE;
System.out.println(intmax);  //output is   2147483647
```

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:

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

- Example of potential solution:

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

Changed from int to long

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:

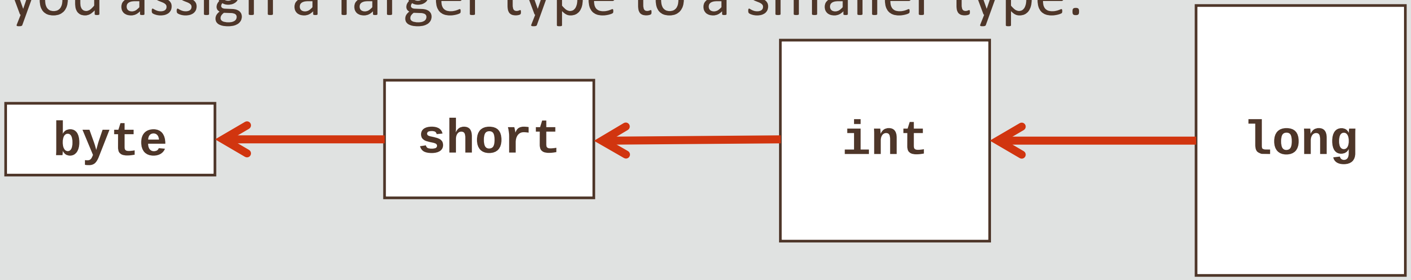
```
int num1 = 7;
double num2 = 2;
double num3;
num3 = num1 / num2;          //num3 is 3.5
```

Changed from int to double

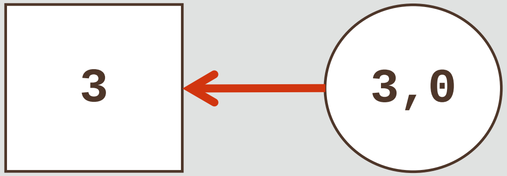
ORACLE
Academy

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;`

Type Casting Example - Rounding

- Type casting is useful when we need to round numbers
- For example:

```
double a = 1.4;
System.out.println((int)(a + 0.5));  //output is  1

double b = 1.8;
System.out.println((int)(b + 0.5));  // output is  2

double c = -1.4;
System.out.println((int)(c - 0.5));  // output is  -1

double d = -1.8;
System.out.println((int)(d - 0.5));  // output is  -2
```

Type Casting Example - Rounding

- Can you predict what will be the results of the following four type casting statements?
- Are the results correct for rounding ?

```
double a = 12.4;
System.out.println((int)(a + 0.5));

double b = 12.8;
System.out.println((int)(b + 0.5));

double c = -14.4;
System.out.println((int)(c - 0.5));

double d = -14.8;
System.out.println((int)(d - 0.5));
```

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
```

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
```

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 (+/-)

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`

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
```

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);`

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

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
```

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;
```

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

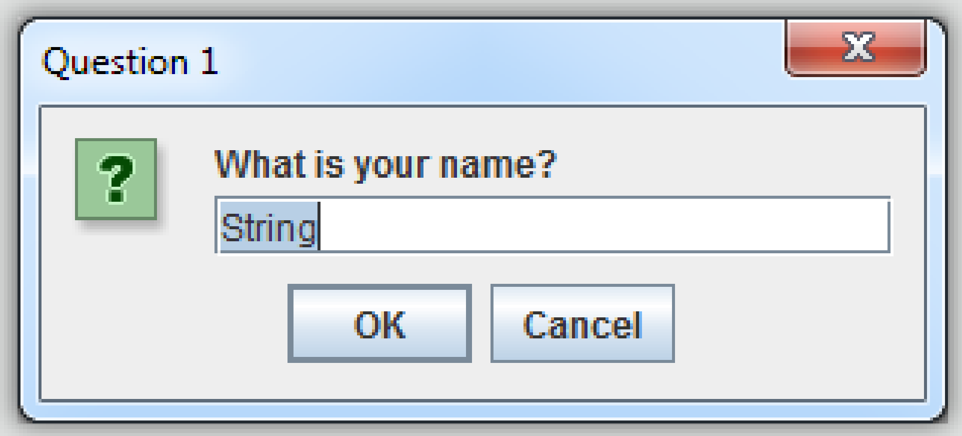
- You may have noticed the underscores (_):
  - As of Java SE7, you can include underscores when you assign numeric values
  - Underscores help large numbers become more readable
  - Underscores don't affect the value of a variable
- The following two statements are equivalent:

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

```java
int x = 123456789;
```

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



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");
```

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

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